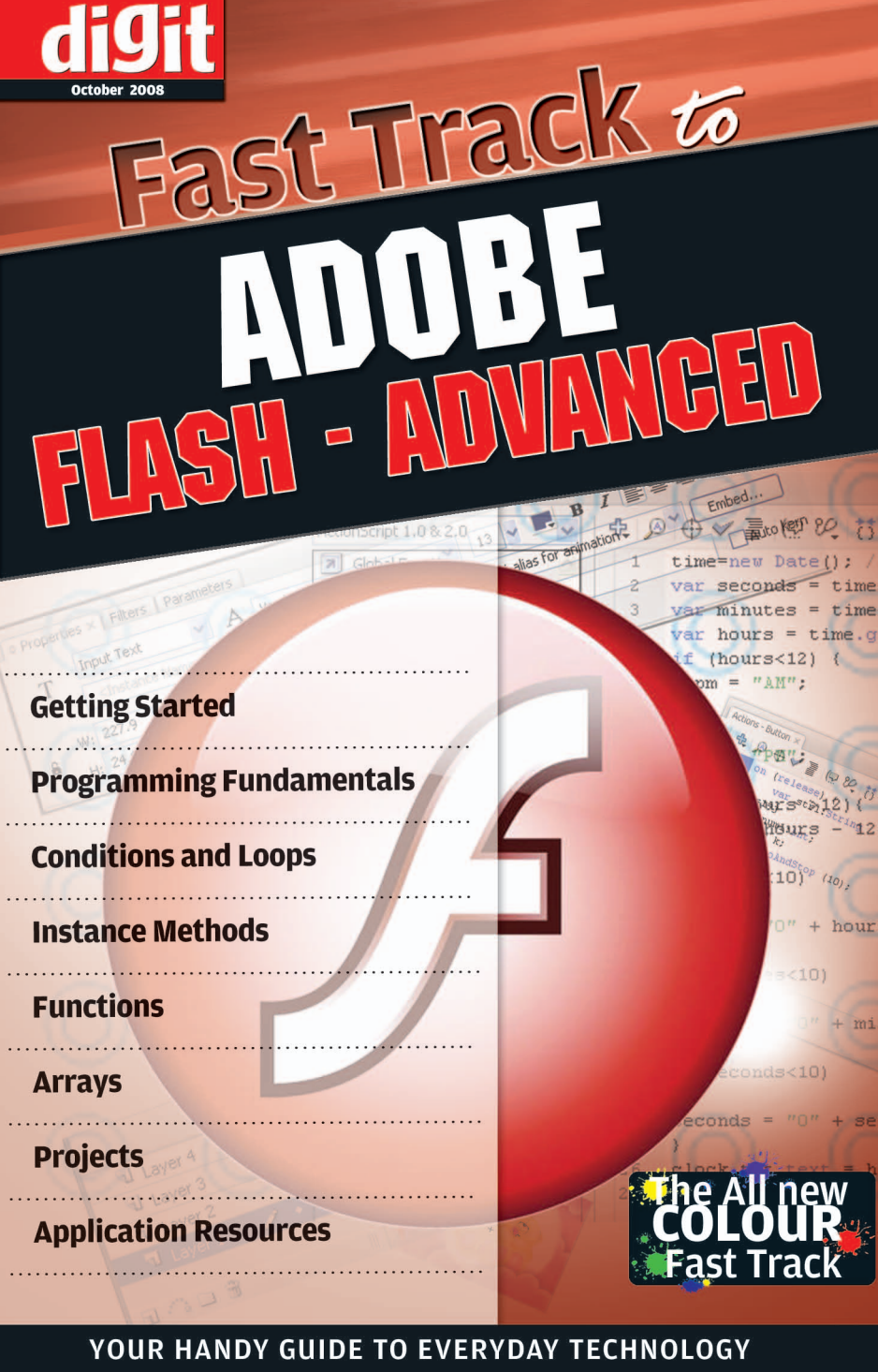
digit
October 2008

Fast Track to ADOBE FLASH - ADVANCED



- Getting Started
- Programming Fundamentals
- Conditions and Loops
- Instance Methods
- Functions
- Arrays
- Projects
- Application Resources

The All new COLOUR Fast Track

Fast Track to Adobe Flash Advanced

By Team Digit

Credits

The People Behind This Book

EDITORIAL

Editor-in-chief	**Edward Henning**
Assistant Editor	**Robert Sovereign-Smith**
Writers	**Santanu Mukherjee, Supratim Bose, Nilay Agambagis Soumita Mukherjee, Ranita Mondal**
Flash Resource Development	**Runa Chowdhury**

DESIGN AND LAYOUT

Layout Design	**Vijay Padaya, U Ravindranadhan**
Cover Design	**Rohit Chandwaskar**

Published by 9.9 Interactive

October 2008

Free with Digit. Not to be sold separately. If you have paid separately for this book, please e-mail the editor at **editor@thinkdigit.com** along with details of location of purchase, for appropriate action.

Action Scripting

Adobe Flash (previously Macromedia Flash), brings with it a unique scripting language called Action Script, which allows developers to do virtually anything with Flash applications and Web content. For example, *Digit's* DVD and CD interface is based on Flash Action Script, and as most of you can see, Action Script allowed us to build a virtual OS inside our discs, to make it easier for you to browse through the contents.

This *Fast Track* will pick up from where we left off last month, and take you through basic Action Script principals, and focus more on Flash programming.

To help you understand the workshops that are in this *Fast Track*, we have included some sample Flash files that you can open with Adobe Flash and look at the code involved. You will find these files on this month's Digit DVD.

We will walk you through some programming fundamentals in Chapter 2, so that you have your basics right. We move on to show you how to use conditional statements, and then look at Instance Methods and Functions, to help you write your first blocks of Action Script code.

We take things to a little higher level next, by looking at Arrays in Action Script, which will help you manage your data better for your coding purposes.

It's not all theory either, and we also have two actual Projects, where you can learn some practical skills through the exampes provided. Our student readers are sure to love this section, because a lot of these examples will help you build better projects—for those who choose to do a project in Adobe Flash that is.

The second project will help you understand how our DVD and CD interfaces are made, and you can now make your own disc interfaces for any project that you have to work on. Finally, the last chapter will help you add in a lot of fun aspects to your Flash creations.

Let us know how much this Fast Track helped you by sending feedback to *editor@thinkdigit.com*.

CONTENTS

CONTENTS

Getting Started

1.1 Difference between AS2 and AS3

Before going through the differences between ActionScript 2.0 and ActionScript 3.0, let us look at the basics of ActionScript. Without an overview of the basic ActionScript it will be difficult for us to understand the differences.

ActionScript is a scripting language that follows the Ecma (European Computer Manufacturers Association) Standard. This scripting language is used while developing either a website or software using Adobe Flash Players. ActionScript is the line of instruction that we insert in the Actions panel of Flash for the execution of some actions. We can make a webpage or software interactive by inserting ActionScript commands. Suppose you want to create a form that is to be filled in by the users. You may want to insert a 'Submit' button so that users or visitors to your Website can submit the form online. To do this you can insert ActionScript in the Actions panel of Flash.

The ActionScript that you insert in the Actions panel is executed by the AVM, i.e., the ActionScript Virtual Machine. This AVM is an inseparable part of the of Flash Player system. The ActionScript codes are written in a bytecode format — a programming language format. These codes are usually placed in an SWF file that can be executed by Flash Players.

When Flash Player were updated from Flash Player 2 to Flash Player 9 the version of ActionScript was also updated. There are three versions of ActionScript: ActionScript 1.0, ActionScript 2.0 and ActionScript 3.0.

The Actions panel is the container where we can insert the scripts. Now let us quickly go through the features of the Actions panel:

The Actions panel includes the following attributes:

- **Toolbox**, from where we can easily access various options, like,

Add a new item to the script, Find, Insert a target path, Check syntax, Auto format, Show code hint, Debug etc. In a later section we will describe each of these tools in detail.

● **Script Category**, where we can get a drop down menu of various versions of ActionScript (ActionScript 1.0, ActionScript 2.0 and ActionScript 3.0) and Flash Lite (Flash Lite 1.0 ActionScript, Flash Lite 1.1 ActionScript, Flash Lite 2.0 ActionScript, Flash Lite 2.0 ActionScript, Flash Lite 2.1 ActionScript).

● **Script Assist button**, by clicking this button we can make a wizard appear on the screen, so that you can then select required attributes from the available menus.

● **Help button**, by clicking this button we can easily access the Help window of Flash CS3. Here we can easily get our required information not only about ActionScript but also about the Flash software.

● **Options menu**, which is an exhaustive list of some supplementary settings.

● **Pin active script icon**, by clicking this icon we can include a tab for a specific ActionScript. If we click on this icon then a window will appear on the screen, keeping that script available even when we move away from the relevent frame or object — it is kept active.

● **Close pinned script icon** that will only appear if we click on the Pin active script. The name itself suggests that we can close the Pin active script window by clicking on this icon.

● **Current script field**, that will displaythe script data that is presently being worked with. You can easily view information about the present script if you place the mouse cursor on this field.

● **Navigator Window**, where you can view all the scripts that are inserted into the animation.

● And last but not the least, the **Script window** where you can view the ActionScript that you will insert into the Flash movie. These ActionScripts appear in the order of execution of the script.

Now you should have got at least some idea about the basic s of ActionScript and are well prepared to understand the basic differences between ActionScript 2.0 and ActionScript 3.0. So without delay let's look at the following:

We have just described that ActionScript 1.0 is the oldest version of ActionScript and ActionScript 3.0 is the latest. At the very beginning we need to know how this newest version of ActionScript is different from the earlier versions of ActionScript. ActionScript 3.0 requires a brand new virtual machine to execute the script. As ActionScript 2.0 is the most commonly used scripting language for Flash, the scripts of ActionScript 2.0 can be published in more than one versions of Flash. ActionScript 3.0 is not used by many users and the scripts of the ActionScript can only run in the Flash CS3 version.

In the case of mobile phones you can use Flash Lite 2 and Flash Lite 3; ActionScript 2.0 is the basis of both Flash Lite 2 and Flash Lite 3. You can easily load the external SWF files that are coded with ActionScript 2.0. ActionScript 3.0 can load the external files that are coded with ActionScript 2.0. Users who are already having websites that are designed with ActionScript 2.0 often feel uneasy of convert ing their web pages to ActionScript 3.0, as it will be very difficult for them to edit the pages. Besides all this, ActionScript 3.0 editors are not as readily available as ActionScript 2.0 editors. So people always feel safer editing their web pages using ActionScript 2.0.

1.2 Tools for Writing AS code

In the earlier section we have already mentioned the Tool box of the Actions panel. These tools are the basic tools for writing ActionScript in Flash. Now let us look at these tools:

- Add Statement
- Find
- Insert Target Path
- Check Syntax
- Auto Format

- Show Code Hint
- Debug Options
- Collapse Between Braces
- Collapse Selection
- Expand All
- Apply Block Comment
- Apply Line Comment

Now let us go through the details of these Tools:

Add Statement

If you click on this Add Statement button a drop down menu will appear. This drop down menu includes various language elements. Each of the listed items includes many sub-options. This panel is powerful and exhaustive, and we will see that each of these sub-options includes many other sub-options. Some of the short cut keys are also listed with these options. Here what we need to do is to select our desired element in order to add it into our script.

For example, say you want to insert a script for a 'play' button into your FLA file. Here you need to click on the Add Statement button. From the drop-down menu that appears, you can select the Global Functions option. Again, you will see that a new drop down menu will appear. From this drop-down menu you then need to select the Timeline Control option. Here a new options list will appear. From this drop-down menu you can select the Play option. You will see that the short-cut key for this Play button is also given here and this key is: Esc+pl.

Find

This is the Find and Replace option of the Actions panel. If we place our mouse cursor on the button the option name will be shown as 'Find'. The function of this Find option is just the same as the Find and Replace option in other software. If you click on this button the Find and Replace dialog box will appear on screen. Here you need to insert the text/code that you want to find in your inserted script. Then in the Replace field you can insert the script that you want to place here.

Insert Target Path

You can address some specific objects by using this Insert Target Path button. At the same time by using this button you can specify a target clip. If you click on this button you will see that the Insert Target Path dialogue box will appear. This Insert Target Path dialogue box includes a pane and two buttons: Relative and Absolute. After inserting the Target Path you can click on the Ok button or Cancel it by clicking on the Cancel button.

Check Syntax

This option is used to check for the errors that occur in the syntax of the current script. These syntax errors appear in list form in the Output panel.

Auto Format

Say you have inserted perfect code in the Actions panel but you can not execute the code. You have clicked on the Check Syntax button to check whether you have inserted any wrong code. But here also the wizard says that there is no error in the syntax. Then someone tells you that the formatting is not perfect in the code that you have inserted there. So you click on the Auto Format button and Flash formats the code automatically. It can add proper indentation to your code, and thereby improve the readability of the code. You can also set the Auto formatting option in the Preference dialogue box. To reach this Preference dialog box you just need to select the Preference option from the Edit drop-down menu. Here the Preference dialog box will be opened. In the Category you will see the Auto Format option. Just click on this option and a new window will then open. Here you can check on the option boxes to insert necessary formatting attributes. Finally, to save the changes as well as close the dialogue box, click OK.

Show Code Hint

This is also an extremely helpful option. It can help you to complete your ActionScript in the Actions panel. While scripting you may need to get assistance from some wizard that can provide you some code hint and where from you can choose and use your

desired code. While developing a page in the Code view of Dreamweaver 8 you can open the tag hint list and from there you can select and use the required code. Similarly, here you can also select and use code from the available wizard. If you uncheck the automatic code hint option then you can get some help from the Show Code Hint option that will display a code hint on your screen.

Debug Options

This is also a wonderful option button that enables you to add and remove breakpoints where you want Flash to pause at a particular line of code in order to investigate the way it is playing. This is required in cases when you feel that things are not going according to expectation. At the same time you can delete these breakpoints easily. What you need to do is just to click in the gutter to the left side of a line of code. A red dot usually appears to the left of line of code that we are working on.

Collapse Between Braces

'Code folding' of the parts of the script can be triggered by clicking on the Collapse Between Braces button.

Collapse Selection

Perhaps whilst scripting you often find that your script is becoming so huge that you have difficulty in handling it. Here, if you click on this Collapse Selection button you can hide the parts of the script that you have already completed satisfactorily.

Expand All

Say you want to view code that you have hidden by using the Collapse Selection option. You may need to change or modify that code. In that case you can click on the Expand All button. This will make the hidden code reappear in the Script Window. The same function can be performed in other ways. When you select a particular section of code, a '+' or '-'sign appears in the left margin. You can click and either hide or show the selected parts of the code.

Apply Block Comment

You can insert a note or comment for yourself in the code. You may need to write something for future use. In any language, this is called commenting. These comments are completely ignored by Flash. You inserts comments between /* and */ signs. You can also add multiple lines of comment by clicking on this Apply Block Comment button.

Apply Line Comment

As we have discussed in the Apply Block Comment section you can add multiple line comments by using the Apply Block Comment button. In case you need to add a single line comment, you can use the Apply Line Comment button. Here, you need to insert the single line comment within a double slash (//) sign at the beginning of the comment. These comments can easily be identified as the text appears in grey colour in the script.

1.3 Concept of Oops

OOP is the acronym for Object Oriented Programming — this is the basic model used for scripting languages. Objects and their interaction within the program are the basis of the OOP model. When it was introduced in the 1960s it was not supported by many programming languages, but things started to change in the early 1990s. At the present time many programming languages support this model. Xerox PARC first coined the Term OOP in order to describe a computer application that was designed using objects. The popularity and acceptability of object-oriented programming languages started to rise in the 1980s. At present, the popularity of programming languages in the OOP category is unquestionable. Python, .NET and JavaScript are among such scripting languages.

The reason for this popularity is that OOP offers greater flexibility than other programming language styles.

The basic concepts of this type of programming language are as follows:

- Classes
- Objects
- Methods
- Encapsulation
- Polymorphism
- Inheritance

Now let us go through the details of these features:

Classes

Classes are the basic outline of an object. When programming with an object-based programming language you can easily overlook all those concerns that are not directly related with the behaviour of an object or with the state of that object. This suggests that while programming we need to provide an outline of an object. We also need to keep in mind that the class does not stand for an object. We can consider the class as something like an extensive type statement.

Let us go through an example of Class:
The following 'Car' class is one possible implementation of a car:

```
class Car {

        int tempo = 0;
        int velocity = 0;
        int gear = 1;

        void changeTempo(int newValue) {
              tempo = newValue;
        }

        void changeGear(int newValue) {
              gear = newValue;
        }

        void velocityUp(int increment) {
              velocity = velocity + increment;
```

```
        }

        void applyBrakes(int decrement) {
              velocity = velocity - decrement;
        }

        void printStates() {
              System.out.println("tempo:"+tempo+"
velocity:"+velocity+" gear:"+gear);
        }
   }
```

Objects

The most basic feature of OOP is the object. This object is nothing more than a combination of some variables and some methods that are related to those objects. In most programming languages scripts are written irrespective of references to the real world. In the object-based programming model we can include reference to the real world in the programs. Here the functions are nothing but the representation of the facts of day-to-day life in the programs.

Methods

In a programming language the methods are the actions that affect only one specific object. In a practical example we can say that the main actions are defined as methods. For example, drink(), sleep(), dream() etc. would be called methods.

Say Jack is sleeping. Here Jack is a class and sleep() is the method associated with Jack.

Encapsulation

Encapsulation is a system to bind code and data together. By using this process we can keep both data and code safe. Here we can easily avoid meddling and misuse of data and code. In short we can say that we can separate some particular code and data from the other code and data by using this encapsulation. In other words we can say that encapsulation can hide the functional details of a particular class from the objects that are used to send messages to it.

In the above section we have described a class, i. e, Jack and the associated method, sleep(). The code for the sleep() method defines all the details of the method (how Jack sleeps, etc). When encapsulated, this information cannot be revealed to others. We can protect this information from other classes by using the system of encapsulation. By using this system we can easily edit our web pages where the clients' status will not be changed as some certain data will always be hidden and protected from the clients.

Polymorphism

Polymorphism is a unique feature of OOPs. By using this feature we can use a particular interface for a generalised class of actions. The exact nature of the situation can determine the specific action. In a nutshell we can say that polymorphism is a system that includes a single interface and multiple methods. We can design a general interface to a group that includes multiple activities. Many complexities can be lessened as here we can apply a single interface for multiple actions. The 'method' can specify the actions that are to be applied via the common interface.

Inheritance

Inheritance delivers us a strong and powerful system to structure our desired software. The classes of the programming language usually inherit their state and behaviour from super-classes. Inheritance is nothing but the derivation of one class from another class.

Let us explain this inheritance a bit:

Nowadays, a single object or a commodity is frequently updated in the course of time. At the same time a lot of variation of a single object is also available due to enormous market competition. Say you want to club some objects that are similar to some extent or that are nothing but the variation of a single object. So here by applying the theory of inheritance you can define the parent object as the super-class and the variations of the parent object as the sub-class.

1.4 Creating a basic "Hello World" Program

Now let us go through the Hello World application. This Hello World is a single line text that is displayed on screen. It uses a single object-oriented class that is named as Greeter. This Greeter can be used from within a Flash document. A Greeter can also be used in a Flex application.

Our first task is to create a basic version of a Flash application. Once it is created, our next target will be to include some new functionality by using which the user can insert a user name, at the same time the inserted user name can be checked against a list of other available users.

Now let us see how we can create a Hello World program using the ActionScript:

First you have to click on the File menu. Here a drop-down menu will appear and from this drop down menu you can select the New option. This will open the New Document dialogue box. You then need to select the Flash File (ActionScript 2.0/3.0) and then click OK. A new Flash document will then be displayed.

Again, you need to click on the File option and then select the Save As option from the drop-down menu that appears. The document will then be saved with your desired file name. Say you save this document with the name HelloWorld.fla. Here .fla is the file extension for Flash CS3. In the Flash Tools palette you then have to select the text tool and you must drag this across the Stage so that you can define a text field.

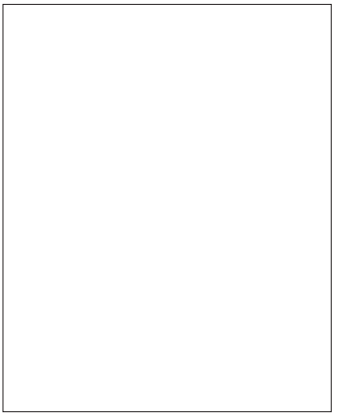
This text field needs to be approximately 300 pixels wide and 100 pixels high. Next, in the Properties window, just type mainText as the instance name of the Text Field. Now you should navigate to the Timeline and here you have to click on the first frame of the Timeline. You then need to open the Actions panel. Here you have to click on the Window option and from the Window drop down menu you can select the Actions option — this is needed for typing the code.

Lets us learn how we can display the string "Hello World" using Action Script. Follow these steps for creating the application:

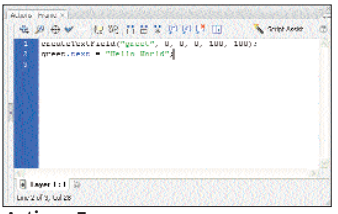
- Open a Flash Document. If the new document needs to be opened in Flash Actionscript 2.0, it should be set here
- Right Click on the 1st key frame
- Choose Actions
- Now type the following code

```
createTextField("greet",
0, 0, 0, 100, 100);
    greet.text = "Hello
World";
```

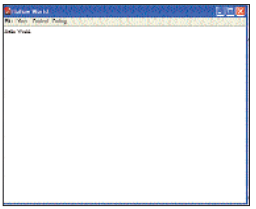
- With this code a text area will be created with depth 0 and position (0,0)
- The dimension of the created Text field will be 100,100 (pixels)
- In that text field the string Hello World will be displayed
- Press [Ctrl] + [Enter] to run the application



Actions Window



Actions Frame



Hello World window

Programming Fundamentals

2.1 Types of Data

Different data types are available in ActionScript to be used for variables created by the programmer. Some of the data types that are used are classified as fundamental data types. The major two data types that are found in Flash ActionScript are the String and Numeric data type. Now let us discuss these data types in more detail.

String: The String data type represents a textual value, like a name of a person. A few examples of the string data type are as follows:

- "Adobe "
- "Let us learn Action Scripts"
- "1200"

Numeric: In ActionScript the numeric data type mainly consists of four specific data types which are number, int, Uint and Boolean. Now we'll discuss these data types in more detail.

Number: The number data type represents a numeric value which may include both integers (whole numbers) or floating point (numbers including a decimal part).
Eg: 123, 9.537 etc
Int: The integer data type which is a whole number without any decimal part
Eg: 86, -28 etc
Uint: The Uint data type is related to an unsigned integer which denotes a whole number that cannot be negative
Eg: 100, 46 etc
Boolean: This data type has two values: true or false.

In ActionScript, simple data types represent a single piece of information: these may be a single number or a single sequence of text but it is to be noted that the majority of the data types defined

in ActionScript can be described as complex data types as they represent a set of values which are grouped together. For instance, if we consider a variable having a data type such as Date, it will hold a single value.

Though 'date' is considered as a single value by declaring and creating a Date variable, internally the computer will consider it as a group of several values that, put together, define a single date. Most of the built-in data types that are found in Flash Action script are basically complex data types. Some of the most important complex data types that are found in ActionScript include:

MovieClip: This data type is related to the clip symbol.
TextField: The TextFileld data type is frequently used in ActionScript which is a dynamic and input text field.
SimpleButton: This data type is related to a button symbol.
Date: The Date data type gives suitable and relevant information about a single moment in time.

Another two prominent and frequently used words that are used as synonyms for data types are the 'class' and 'object'. A class is basically used when we define a data type. On the other hand, an object is defined as the instance of the class. Let us take an example where the same thing has been expressed in three different ways.

The data type of the variable myName is a Number.
The variable myName is a Number instance.
The variable myName is a Number object.
The variable myName is an instance of the Number class.

It is to be noted that variables in flash have the capacity to accept any type of data at any time even if the variable earlier had a numeric value. It is also possible to overwrite a different data type. For instance:

```
var myName;
myName = "Jack";
```

2.2 Variables

A very common concept that is found in most scripting and programming languages is that of the variable. This can be considered as a container that holds data and can be used later on when necessary. In Flash ActionScript a letter or any particular word is assigned to identify such a variable. The user can also assign numeric values and characters to it. A variable is primarily used to store values, and the var statement is mainly used for declaring those variables. For example:

```
var x; // a variable x is declared
```

If the var statement is omitted while declaring a variable, a compilation error in strict mode and run-time error in standard mode will be displayed. For example consider the following statement:

```
i; // error will occur if i was not previously
defined
```

On the other hand, when the user wants to declare a variable of a particular data type, say an int, the following declaration must be given:

```
var x:int;
```

Besides this, a variable can also be assigned a value by using the (=) operator. For example, the following code declares a variable k and assigns the value 20 to it:

```
var k:int;
k = 20;
```

The user can also declare and initialize an array or instantiate an instance of a class. The following code shows an array that is declared and is also assigned relevant values.

```
var numExample = ["zero", "one", "two"];
```

Besides the above method, you can also create an instance of a class with the help of the new operator. The following code creates an instance of a named CustomizedClass, and assigns a reference to the newly created class instance to the variable named customerItem:

```
var customerItem:CustomizedClass = new CustomClass();
```

While declaring more than one variable, you can declare them all in a single line of code with the help of the comma (,) operator and assign values to them as shown below:

```
var x:int = 30, y:int = 20, z:int = 30;
```

Understanding variable scope:
The scope of a variable is a description of the accessibility of that variable. There are mainly two types of variables that are used in Flash ActionScripting. One is the global variable that is accessible from anywhere in the entire code; while the local variable is one that is defined in only one part of the code, and only accessible from there. A global variable can also be defined outside any function. The following code shows the definition of a global variable, available from both inside and outside of a function.

```
var strGlobal:String = "INA";
function scopeTestFinal()
{
    trace(strGlobal); // Global
}
scopeTestFinal();
trace(strGlobal); // Global
```

If the variable name that is used for a local variable is already declared as an global one, the global definition is hidden by the

local definition while the local variable is in scope.

Unlike in other programming languages (C or C++), the variables used in Action Script do not have a block-level scope. A variable declared inside a block of code is accessible not only to that block but also to any other parts of the function related to that block. The following code snippet contains variables that are defined in various block scopes. All the variables are available throughout the function.

```
function blockTest (testingArray:Array)
{
     var numElement:int = testingArray.length;
     if (numElement > 0)
     {
          var elemStr:String = "Element #";
          for (var i:int = 0; i < numElement; i++)
          {
               var valueStr:String = i + ": " +
testingArray[i];
               trace(elemStr + valueStr);
          }
           trace(elemStr, valueStr, i);    // all
still defined
     }
     trace(elemStr, valueStr, i); // all defined
if numElements > 0
}

blockTest(["Earth", "Moon", "Sun"]);
```

Default Values

The default value of a variable is said to be that value which a variable contains before a new value is assigned to it. A variable declared but without a value is said to be in an uninitialised state whose value further depends on its data type. The table provided below gives the default values of variables organised according to their data types.

Data type	Default value
Boolean	false
int	0
Number	NaN
Object	null
String	Null
Uint	0
Not declared	Undefined
All other classes	null

2.3 Creating Objects

Objects are the most important components in Flash ActionScripting. They are basically complex values containing additional values or variables known as properties. An object can be created by using the new operator. The following code creates a new object instance.

```
// Creating a new object instance

var myValue:Object = new Object();
person1 = new Object();
```

The above code creates two object instances, called myValue and person1.

Adding Properties

Properties can be added to the object instances with the help of the .operator as shown below:

```
person1.firstName = "Frodo"; // here a property,
firstname is added to the object instance person1.
```

Accessing Properties

For accessing the properties of an object, the programmer can do this using the dot operator or the same can be achieved by using

an array, as shown below

```
trace( person1.firstName );
trace( person1[ "firstName" ] );
```

Adding Methods

In Flash Action Scripting, a variable can be assigned to a function which can then be called indirectly by using that particular variable. This is illustrated below.

```
function mySystem( msg1 ) { trace( msg1 ); }
iFunc = mySystem; // assign function to variable
iFunc( "Hello " ); // make indirect call
```

For adding a method to an object, the property is added to the object instance with the help of the (.) operator as shown below.

```
person1.sayName = function() {
trace( this.firstName );
}
person1.sayName(); // call the function
```

Object Destruction:

Flash Action Scripting can handle garbage collection — i.e. it does not explicitly delete objects when the program terminates as the objects are deleted automatically by using the delete keyword. This is shown below.

```
delete person1; // the object instance person1 is deleted by the delete keyword
```

Sometimes instead of deleting the whole object, we can also delete parts of the object as described below.

```
delete person1.firstName; // here only the property firstname is deleted and not the whole object.
```

2.4 Namespace, Operators and Type Checking

Namespace

In Flash ActionScripting, namespace plays a vital role in control-

ling the visibility and properties of the methods that are created by the programmer. This helps with the control of access specifiers. However it is to be noted that we can create our own namespaces.

Before understanding the working of a namespace, one should remember that the name of a property or a method always contains two parts, the identifier and a namespace. Consider the following code segment.

```
class Sample
{
     var sampleGreet:String;
     function sampleFunction () {
                    trace(sampleGreet + " from
sampleFunction()");
     }
}
```

When a definition is not preceded by a namespace attribute, the names are qualified by a default internal namespace. This implies that they are visible to callers belonging to the same package. In the code snippet above, both the sampleGreet and sampleFunction () have an internal namespace value.

Operators

In Flash ActionScripting, operators play a major role. They are basically special functions that work on one or more operands and return a suitable value. The value may be a literal, a variable or an expression that is used as an input by the operator.

For instance, take the following code. Here the addition (+) and the multiplication (*) operator are used with three literal operands (2,3,4) to return a value which is further used by the assignment operator to assign the returned value to the variable sumNumber.

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

Various types of operators are used in Flash ActionScripting. The most common operators that are used include the unary, binary or the ternary operator. In the case of the unary operator, one operand is taken (the increment or ++ operator) while in the

case of the binary operator, two operands are need (such as the division operator). On the other hand, the ternary operator works on three operands (e.g. the conditional operator).

Besides these, there are operators in ActionScript which are overloaded — i.e they behave differently depending on the type or number of operands passed to them. The most common example of an overloaded operator is the addition operator. The addition operator returns the concatenation of two operands when both the operands are strings. This is illustrated below.

```
trace(5 + 5);       // 10
trace("5" + "5"); // "55"
```

Besides these operators, other commonly used operators that are used in Flash ActionScripting are:

- Primary operators
- Postfix operators
- Unary operators
- Multiplicative operators
- Bitwise shift operators
- Relational operators
- Equality operators
- Bitwise logical operators
- Logical operators
- Conditional operator

Type Checking

Type checking is an important property that is used extensively in Flash ActionScripting. It can occur either during compile time or during run time. Flash ActionScripting support both run and compile time type checking. The type checking is performed with the help of a special compiler mode called strict mode in which type checking is done both at compile and run time. On the other hand , in the case of standard mode, type checking occurs only at run time.

Compile-time Type Checking

In the case of compile-time type checking, it is advisable that it should be used in the case of larger projects, as in such projects it becomes more important to catch type errors than to concentrate on data type flexibility. It is for this reason that the compiler used in Flash ActionScripting is set to run in strict mode.

Consider the following code snippet. Here the code adds data type information to the xParam parameter, and declares a variable myParam with an explicit data type:

```
function runtimeTest(xParam:String)
{
   trace(xParam);
}
var myParam:String = “hello”;
runtimeTest(myParam);
```

Run-time Type Checking

Whether a program is run under strict or standard mode, run type checking is always executed. The following example shows a function named typecast which expects an array argument but receives a value 3. Parsing of this value generates a run time error in standard mode as 3 does not belong to the parameters declared for data type Array.

```
function typeCast()   (xParam:Array)
{
     trace(xParam);
}
var myNum:Number = 3;
typeCast()     (myNum);   //   run-time   error   in
ActionScript standard mode
```

2.5 Built In Classes

In Flash ActionScript, class plays a major role, as every object is defined by it. Class is often thought of as a template for an object

type and includes data values and methods. The values stored in the properties are mainly the primitive values which may be numbers, strings or Boolean values.

Various built-in classes are present in Flash ActionScript.The most common built in classes that are found include Number, Boolean and the String classes which mainly represent primitive values. Other built-in classes such as the Array, Math and XML classes define complex objects which form part of the ECMAScript standard. However in ActionScript 3.0, a new concept of untyped variables was introduced which can be defined in the following two ways:

```
var someObj:*;
var someObj;
```

A unique feature of the untyped variable which makes it different from the typed one is that the untyped variable has the capacity of holding special values that are undefined. Classes can also be defined by using the class keyword while the class properties can be declared in three ways — i.e. by using the constant keyword for declaring constants, var for declaring variables and get and set methods for defining getter and setter methods. On the other hand, an instance of a class can also be created with the help of a new operator. This can be demonstrated with the following code.

```
var myBirth:Date = new Date();
```

2.6 Packages

In Flash Action Scripting, packages play a pivotal role; these are implemented along with namespaces but are not synonymous with them. While declaring a package a special type of namespace is implicitly created that is only known at compile time. The following code uses the package directive to create a simple package having one class.

```
package samplecode
{
     public class SampleCoding
     {
          public var sampleGreeting:String;
          public function sampleFunction()
          {
                    trace(sampleGreeting + " from
sampleFunction()");
          }
     }
}
```

The class that is declared above is the SampleCoding class lying inside the samples package. Due to this the complier automatically assigns the class name during compile time into its qualified name.

2.7 Constructor Methods

In ActionScripting, the constructor methods are sometimes known as simply constructors or functions which share the same name as that of the class in which they are defined. A code that is included in a constructor method is executed whenever an instance of a class is formed with the help of the new keyword. This is explained below where a simple class is created containing a single property. The initial value of the property is always set inside the constructor function.

```
class Examples
{
     public var status:String;
     public function Example()
     {
          status = "initialized";
     }
```

```
}

var myExample:Examples = new Examples();
trace(myExample.status); // output: initialized
```

The constructor methods can only be declared as public and no other access specifiers like private, protected or internal can be used on a constructor. Besides these, an user defined namespace also cannot be used with a constructor method.

For making an explicit call to the constructor of its superclass, a constructor can use the super() statement; otherwise the compiler automatically inserts a call before the first statement in the constructor body. The methods belonging to the superclass can also be called by using the super prefix as a reference to the superclass.

2.8 Creating a Simple Flash Application using basic programming concepts

Example 1:

Swapping of Two Values (.fla file given : Swapping.fla)

Say, you have two text fields where a user can input texts or numbers. Perhaps you want that clicking on a button causes the values of two fields to be interchanged. Let's see how we can programmatically develop this using Flash CS3.

i. Open a blank document in Flash CS3. If the flash application is opened in Action Script 2.0, this should be set here.
ii. Now select the Text Tool and create two text fields on the stage

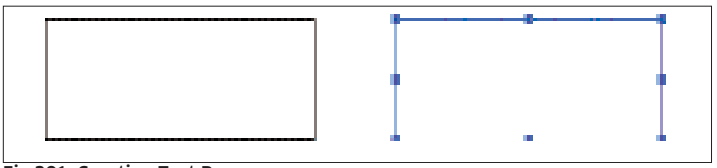


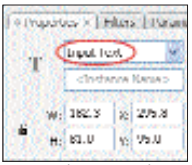
Fig 201: Creating Text Boxes

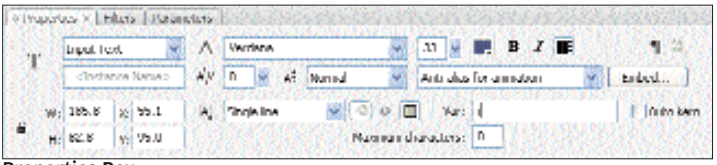
with a suitable font and font size

iii. Set the Text Type of both the Texts as Input Text from the Property Inspector Panel

iv. Set the variable name of the 1st Text field as i and second as j in the Var field

v. Now draw a rectangle in another layer


Property Inspector Panel


Properties Box

vi. Press F8

vii. A dialog box appears on the screen

viii. Select the 'Button ' option

ix. Click OK

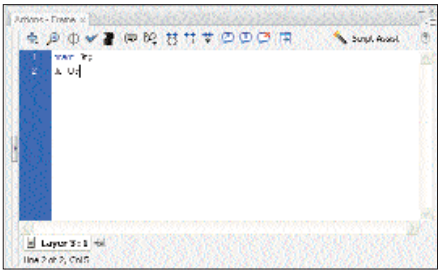

Dialog Box

x. Take a new layer

xi. Now at the 1st frame of the New layer declare 'K' as a variable

xii. To do this you have to add the following script to the 1st Frame and initialize it with 0

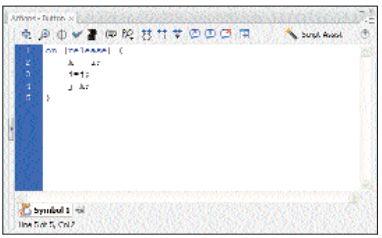
```
var k;
k=0;
```


Actions Frame

xiii. Now right click on the button and choose Action

xiv. Here add the following Script

```
on (release) {
  k = i;
```


Actions Button

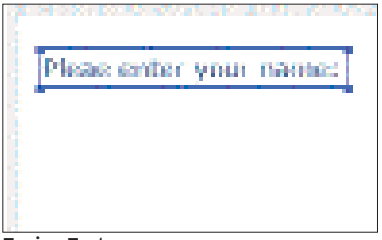
```
    i=j;
    j=k;
  }
```

This means that as you click on the button, whatever is written inside the 1st Text Field will be stored within k. Now the value of i will be replaced by j, the second text field. The initial value of the 1st text field which was i is now stored in k. Now this value will be transferred to j. So at the end the values become swapped.

xv. After you complete the code, press (Ctrl + Enter) to see the effect.

Example 2:

b. Concatenation (.fla file given: Concatenation.fla)



Typing Text

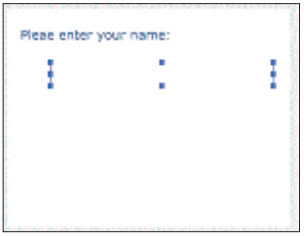
Let us see how we can concatenate one string and one integer data type.

i. Open a Blank Flash File. If the document is opened in Action Script 2.0, it should be set here.

ii. Select the Text Tool and Type "Please enter your name"

iii. Take a new layer

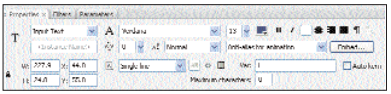
iv. In this layer, with the Text Tool create a Text Box just below the previous text box



Creating a Text Box

v. Set the Text Type as Input Text and set its variable name as 'i' 'j'

vi. To do this you have to type 'j' as Var in Properties Inspector

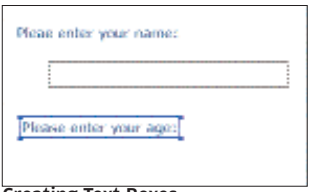

Properties Window

vii. In the same frame of the previous layer "Please enter your age", just below of the previous input text box.

viii. Remember that, here the text type must be normal or Static Text.

ix. Now go to the layer containing the input text.

x. With the Text Tool again create a text box just below the last text box

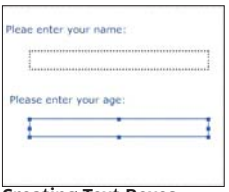


Creating Text Boxes

xi. Here also define the text type as Input text

xii. Set the variable name as 'k' for this input box

xiii. Now create a button to make the movie play



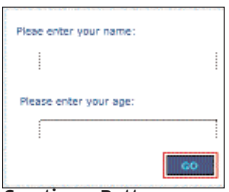
Creating Text Boxes

xiv. Now take a new layer

xv. In the 1st frame of this new layer, add this script:

Stop();

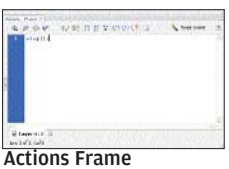
xvi. In the 10th frame add key frame to all the layers.



Creating a Button

xvii. In the 10th keyframe of the Static Text layer, delete all contents

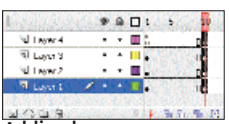
xviii. Also, in the 10th keyframe of the Input Text layer, delete the contents and add a new text field


Actions Frame

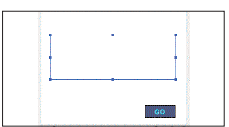
xix. Set the type as Dynamic Text

xx. Set the variable name of this Dynamic Text to finalName

xxi. Now move to the 1st keyframe and right click on the button


Adding layers

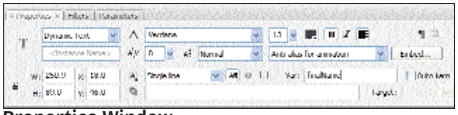
xxii. Choose Actions and type the following code:



Creating a Text Field

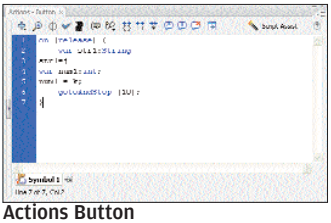
```
on (release) {
     var str1:String
str1=j
var num1:Number;
```

```
num1 = k;
    gotoAndStop
(10);
}
```

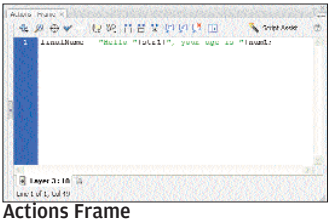


Properties Window

Fig 211 : Actions Button



Actions Button



Actions Frame

xxiii. This means that when the button is clicked then a variable called str1 is declared whose data type is string. Now as "j" is the variable name which can never be a character we will store that inside the string type variable which is str1

xxiv. Similarly num1 is declared as an integer type variable and the variable k, i.e. the age, is stored within it. Remember age is always a number and so we will store it within a integer type variable. The script gotoAndStop(10); basically takes the play ahead to 10th frame.

xxv. In 10th keyframe of the layer containing the script, add another action script:

```
finalName = "Hello "+str1+", your age is "+num1";
```

xxvi. This will cause the 'str1' variable, which holds the name to appear after the word "Hello", and the age, stored within "num1", will appear after the term "your age is".

xxvii. All these will be concatenated as one sentence and the entire thing will be displayed inside the dynamic Text field in the 10th frame.

Conditionals and Loops

3.1 Conditionals

In Flash ActionScripting, three basic conditional statements are used for to control the program flow. The major conditional statements that are used include:

1. If... else
2. If..else if
3. Switch

If .... Else Statement

This is one of the most important conditional statements which helps the user to test a condition and execute a block of code which is dependent on the truth or falsity of the condition. This can be suitably illustrated by the following example.

```
if (x > 20)
{
     trace("x is > 20");
}
else
{
     trace("x is <= 20");
}
```

The above block of code checks the condition i.e x>20. If it's true, then the statement within the curly braces is executed. If the condition is false, the statement within the else part will be executed. For executing an alternative block of code, the if statement without the else part can be used.

If.....Else if statement

For testing more than one condition, the if else if statement can be used. This can be illustrated by the example given below. The code shown below not only tests whether the value of x exceeds 20, but also tests whether the value of x is negative.

```
if (x > 20)
{
     trace("x is > 20");
}
else if (x < 0)
{
     trace("x is negative");
}
```

Switch

Out of the three conditional statements, the switch statement is the most powerful one. This statement can be suitably used when there are several execution paths that are dependent on the same conditional expressions. The switch statement starts off with a case statement and ends with a break statement. This can be described by the following example where the statement prints the day of the week, based on the day number returned by the Date.getDay () method.

```
var sameDate:Date = new Date();
var dayNum:uint = sameDate.getDay();
switch(dayNumber)
{
     case 0:
          trace("Sunday");
          break;
     case 1:
          trace("Monday");
          break;
     case 2:
          trace("Tuesday");
```

```
            break;
        case 3:
            trace("Wednesday");
            break;
        case 4:
            trace("Thursday");
            break;
        case 5:
            trace("Friday");
            break;
        case 6:
            trace("Saturday");
            break;
        default:
            trace("Out of range");
            break;
}
```

3.2 Loops

Various types of loops are also used in flash ActionScripting that allow a programmer to perform a specific block of code repeatedly by using a series of values or variables. The various types of loops that are used in ActionScripting are:

- For
- for..in
- for each..in
- while
- do..while

For loop

In the case of a for loop, the programmer is allowed to iterate through a variable having a specific range of values. The variable is provided with an initial value, a conditional statement for determining the end of the loop and finally the expression that

changes the value of that particular variable. Let us consider one example where the code goes through loop five times with the value of the variable starting with the value 0 and ending at 4.

```
var i:int;
for (i = 0; i < 5; i++)
{
    trace(i);
}
```

For.. In loop

The For.. In loop is frequently used in Flash ActionScipting for iteration based on the properties of an object or the elements of an array. The loop can also be effectively used to iterate based on the properties of a generic object.

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
    trace(i + ": " + myObj[i]);
}
// output:
// x: 20
// y: 30
```

The user can also iterate through the elements of an array by using the For. In loop which is shown below.

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
{
    trace(myArray[i]);
}
// output:
// one
// two
// three
```

For each...In

A very commonly used loop that is used in ActionScripting is the for each... in loop. This loop iterates through the various items of a collection. The items may be the tags in an XML or XMLList object. Consider the follwoing example where a for each... in loop is used to iterate through the properties of a generic object. But unlike the for.... in loop the variable used for each... in loop keeps the value held by the property instead of the name of the property.

```
var myObj:Object = {x:35, y:45};
for each (var num in myObj)
{
    trace(num);
}
// output:
// 35
// 45
```

The for each.. in loop can also be used for iterating through an XML or XMLList object as shown below:

```
var myXML:XML = <users>
  <fname>Jack</fname>
  <fname>Jane/fname>
  <fname>Joseph</fname>
  </users>;

for each (var item in myXML.fname)
{
    trace(item);
}
/* output
Jack
Jane
Joseph */
```

Besides these, we can also use this loop for iterating through the elements of an array. This can be shown with the help of the

following example:

```
var myName:Array = ["six", "seven", "eight"];
for each (var item in myName)
{
    trace(item);
}
// output:
// six
// seven
// eight
```

While loop

One of the most powerful loops, the while loop is used in many applications. This loop mainly acts like an if statement which repeats itself as long as a condition is true. Consider the following example where the code produces the same output as was produced using the for loop. Consider this example:

```
var i:int = 0;
while (i < 5)
{
    trace(i);
    i++; }
```

However there is one major dis-advantage of the while loop over a for loop in that it is easier to create infinite loops using while loops – these cause program crashes.

Do While

Another important loop that is often used in ActionScripting is the do while loop. This loop guarantees that the code is executed at least once and the condition is checked only after the block is executed. The code shown below illustrates the use of a do.. while. The code generates an output even though the condition is not properly met.

```
var i:int = 5;
do
{
     trace(i);
     i++;
} while (i < 5);
// output: 5
```

3.3 Boolean Logic

Boolean logic is an important concept that is used widely in Flash ActionScripting. There are many examples of applications that require multiple factors in branching logic. The two most important and common Boolean operators that are used in Action Scripting are the Logical OR and the Logical And operator.

Logical And Operator (&&)

The syntax that is usually followed while writing logical And operator is as follows:

Usage expression1 && expression2

The above expression returns expression1 if it is declared false or converted to false and expression2 otherwise. The various values that can be converted to false include 0, NaN, null, and undefined. A function call used as expression2 does not call the function if expression1 is evaluated to false.

It is to be noted however that if both operands are of the type Boolean, the result becomes true only if both operands are true. This can be shown by the following table.

Expression	Evaluates
`true && true`	`true`
`true && false`	`false`
`false && false`	`false`
`false && true`	`false`

Consider the following example which uses the logical AND (&&) operator for controlling an application. The test that is performed decides whether the player has won the game or not. Here the two variables that are used, the turns and the score variable, are updated.

```
var turns:Number = 2;
var score:Number = 77;
if ((turns <= 3) && (score >= 75)) {
     trace("You Win the Game!");
} else {
     trace("Try Again!");
}
```

Logical OR Operator(II)

The syntax that is usually followed while writing logical And operator is as follows:

Usage expression1 || expression2

In the above expression, expression1 is returned if it is true and can also be converted to true. On the other hand if expression2 is used as a function call, the function is not called until and unless expression1 evaluates to true. On the other hand, if both the operands are of Boolean type, the result evaluates to true if one or both expressions are true while the result evaluates to false when both the expressions are false. The above statements can be explained by using the following table

Expression	Evaluates
true || true	true
true || false	true
false || false	false
false || true	true

Here is an example that uses the Logical OR (II) operator in an

if statement. Out of the two expressions, the second expression evaluates to true and hence the final result becomes true.

```
var a:Num = 10;
var b:Num = 250;
var start:Boolean = false;
if ((a > 25) || (b > 200) || (start)) {
     trace("the logical OR test passed"); // the
logical OR test passed
}
```

Here a message called "the logical OR test passed". This is because one of the conditions of the if statement (b>200) evaluates to true.

Consider the following example which uses a function call as the second operand and leads to unexpected results. If the expression on the left of the operator evaluates to true, that result is returned without evaluating the expression on the right.

```
function f1():Boolean {
     trace("f1 called");
     return true;
}
function f2():Boolean {
     trace("fx2 called");
     return true;
}
if (f1() || f2()) {
     trace("IF statement entered");
}
```

3.4 Creating A Simple Flash Application Using Loops (fla file given)

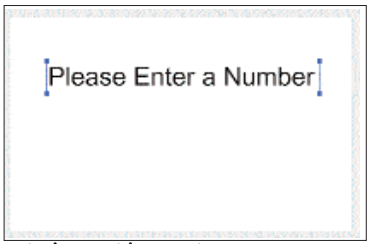
In this unit we have learnt the various concepts of loops and conditional expressions. Now we shall try to develop an application using this concept. A programmer can follow the simple steps that

are given below for creating a Flash file. The file should be saved in .fla format. The steps for creating the application are:

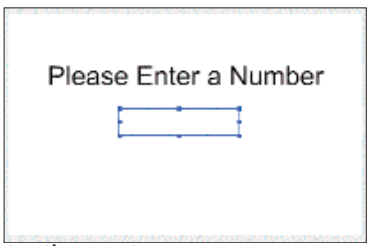
- First open a blank document. If the document is opened in Action Script 2.0 then it should be mentioned here.
- Now select the Text Tool and type "Please Enter a Number"
- Set the Text Type to Static text

Now with the help of the Text Tool draw a text box below it

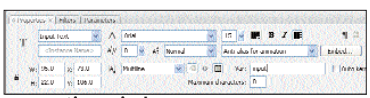
- Set the Text Type of the Text Box to Input Text
- Set the variable name of this input Text box to 'input'
- Now create a new layer
- Here import a smiling face and place it out side of the stage area.
- From File click on Import and then Import to Stage
- Select Smiley. Jpg. (* Note- Here, no image given as a resource)
- Click Open
- Place the object outside the stage
- Select the object... and then press F8 and convert it to a movie clip
- Give the instance name of the Movie Clip as friend
- Now take another layer
- First draw a rectangle just under the input text box
- Select the object... Press F8 and convert it to a button
- Right click on the button and add the following code:



Entering Text in a Text Box

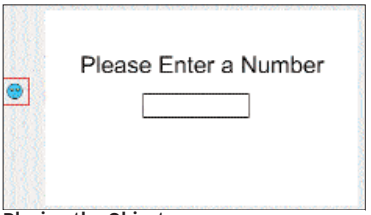


Creating a Text Box

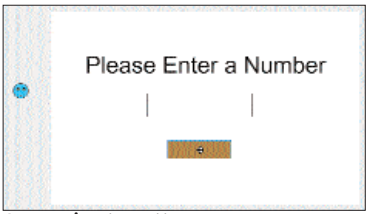

Properties Window

```
on (release) {
for (x=0; x<Number(input); x++) {
```

```
duplicateMovieClip
("friend", "dupMC" + x,
x);
_root ["dupMC"+x]._y
= Math.round (Math. ran-
dom ()*200)
_root ["dupMC"+x]._x
= Math.round (Math. ran-
dom ()*400)
}
}
```



Placing the Object



Conversion to Button

- Press ([Ctrl] + [Enter]) to see the preview

Explanation of the Script

for (x=0; x<Number(input); x++) {

The above code creates a for loop where a variable x is looped the number of times based on the number that is given as input on the input text.

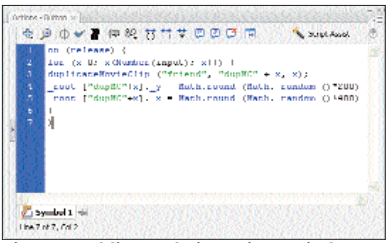


Fig 306: Adding code in Actions Window

```
duplicateMovieClip ("friend", "dupMC" + x, x);
```

This code defines a method named duplicateMovieClip where the movie clip object, friend, is passed as a parameter. This method duplicates the new movie clip and a dynamic and unique name is given to it which was created by adding the value of x to the string dupMC.

```
_root["dupMC"+x]._y = Math.round(Math.ran-
dom()*200)
```

The above code returns a random number which is a multiple

of 200 and also specifies the y position of the movie clip.

```
_root["dupMC"+x]._x = Math.round(Math.ran-
dom()*400)
```

The above code returns a random number which is a multiple of 400 and it specifies the x position of the MovieClip

Instance Methods

Now we will discuss method but the aim will be to focus on the instance method. First, let's see what is meant by method is. Methods are nothing but functions that are included in the class definition. When we create a method associated with an instance then that particular method must go with that particular instance only. We can not use a method outside the instance to which it is attached. We can only do that if we use a function outside a class.

Now let us go through the instance methods in detail. The instance methods are always declared without the static keyword. One thing that we always should remember is that this instance method is always attached to the instances of a class and it is not attached to a class as a whole. These are extremely useful when implementing functionality that can affect individual instances of a class.

If we look at the body of an instance method we will find that the static as well as the instance variables are in scope. This indicates that by using just a simple identifier we can refer to the variables that are defined in the same class.

Example

Let us look at an example of a class, i.e, Customized Array where the Array class is extended. Here a static variable name, i.e., arrayCounting Total is defined by the CustomizedArray class. The total number of class instances can be tracked by this. The arrayNumbering is also an instance variable that can track the order of the creation of the instances. Here the getPosition() instance method can return the values of the variable. Look at the script below:

```
public class CustomizedArray extends Array
{
    public static var arrayCountingTotal:int = 0;
     public var arrayNumbering:int;
     public function CustomizedArray()
```

```
        {
            arrayNumbering = ++arrayCountingTotal;
        }
      public function getArrayPositioning():String
        {
            return ("Array " + arrayNumbering + "
of " + arrayCountingTotal);
        }
   }
```

The external code to the class is referred to the arrayCounting Total, which is a static variable, and it is done through the class object where the CustomizedArray.arrayCounting Total is used. This code is inserted in the body part of the getPosition() method. We can directly refer to the static arrayCounting Total variable. We can also do it in cases of superclasses. The static properties are not inherited in ActionScript 3.0; still we can find that the static properties in superclass are in scope. Let us go through an example. Suppose that the Array class includes a small number of static variables. One of these static variables is DESCENDING. By using a plain identifier the static constant DESCENDING can be referred by the code that resides in an Array subclass.

```
public class CustomizedArray extends Array
{
    public function testStatic():void
    {
        trace(DESCENDING); // output: 2
    }
}
```

The value of the reference that is inserted within the body part of this instance method can also be referred to the instance where the method is attached. Let us go through the demonstrations of this reference that indicates to the instance which includes the method.

```
class Check
{
    function thisValue():Check
    {
```

```
            return this;
        }
    }
    var myTest:Check = new Check();
    trace(myTest.thisValue() == myTest); // output:
true
```

We can control the inheritance of instance by using the keywords override and final. The override feature can be used in order to redefine an inheritance method. We can also use the final attribute in order to stop the subclass from overriding a method.

4.1 Bound Methods

Bound methods are the methods that are taken out from the instance of this method. It is often called method closure. The methods that are passed as arguments to a function as well as the methods that are returned as values from a function are the examples of bound method. As we have already discussed in our earlier section, ActionScript 3.0 includes some new features and the bound method is no exception here. In ActionScript 3.0 a bound method has similarities to a function as it can hold the lexical atmosphere even when you separate it from its instance.

We must now want to know the basic difference between the bound method and the function closure. Let us now look at those differences:

- In a bound method, the 'this' reference for a bound method is always linked with the instance in which the method is implemented. It is always linked with the parent object by which the method is implemented.
- In a function closure the 'this' reference is always nonspecific. It indicates that it can point to any object with which the function is associated at the time when it is called.

We need a detailed understanding of the bound method if we use the 'this' key word. The 'this' key word always gives a reference to the parent object of the method. If we are an ActionScript programmer then we always expect that the 'this' key word will refer to that particular class or object that includes the method's defini-

tion. The instance implementation method is not always referred in ActionScript 2.0. While extracting a method from an instance in ActionScript 2.0 we will see that the 'this' reference is not bound to the original instance. Here we will see that the member variables as well as the method of the class of the instance are also not available. One of the advantages of ActionScript 3.0 is that this problem is not found in ActionScript 3.0. The reason is that the bound method is created while passing a method as a parameter. The fact that the 'this' keyword can always refer to the object or class where the method is identified, is always made certain by the bound method.

Here let us look at the following example of code where the Check class is defined. The Check class includes the case() method that can define the bound method. The external code to the class can generate the Check class and call the block method. At the same time it can store the return value in the myTask variable. Now look at the script below:

```
class Check
{
     private var num:Number = 3;
     function case():void // bound method defined
     {
          trace("case's this: " + this);
          trace("num: " + num);
     }
     function block():Function
     {
          return case; // bound method returned
     }
}
var myTest:Check = new Check();
var myTask:Function = myTest.block();
trace(this); // output: [object global]
myTask();
/* output:
case's this: [object Check]
output: num: 3 */
```

If you look at the last two lines of the above script, you will find

that the 'this' reference in the case() bound method indicates the Check class but here the 'this' reference indicates a global object. The bound method that stores in the myTask variable can easily access the other related variables of the Check class. If you try to run this script in ActionScript 2.0 you will see that the 'this' reference will be the same but the num variable will not be defined.

4.2 Get And Set Methods

By using the get and set method we can hide the basic programming data when we provide a programming interface to our created class to programmers. The programmers can access the properties that are private to the class. We can easily avoid the functions of the traditional accessors that include awkward names. We can also avoid having double public facing functions for a single property. Here the user can get access for both reading and writing.

Look at the example below where the GettingSet class contains the publicAccess() get and set accessors function. This function can provide access to the personal variable personalProperty.

```
class GettingSet
{
     private var personalProperty:String;
        public function get publicAccess():String
     {
          return personalProperty;
     }
                    public     function     set
publicAccess(setValue:String):void
     {
          personalProperty = setValue;
     }
}
```

The following error will appear when accessing the personalProperty in a direct manner:

```
var myGettingSet:GettingSet = new GettingSet();
trace(myGettingSet.personalProperty);   //   error
occurs
```

As an alternative, the user of this GettingSet class can use something that may appear as a property of publicAccess. This is nothing but a pair of get and set accessors functions. This function can operate on the personalProperty.

Here is an example of the GettingSet class that sets the value of the personalProperty. Here we use the public accessor named publicAccess:

```
var myGettingSet:GettingSet = new GettingSet();
trace(myGettingSet.publicAccess); // output: null
myGettingSet.publicAccess = "hello";
trace(myGettingSet.publicAccess);  //  output:
hello
```

We can override the inherited properties from a superclass by using the Get and Set functions. This is not possible when using the regular variables of class member. If we declare a class member variable by using the var key word then it will not be possible for us to override it in a subclass. At the same time we should always keep in mind that this limitation is not applicable for the properties that we create by using the get and the set function. The override feature can be used on the get and set functions but these get and set functions must be inherited from a superclass.

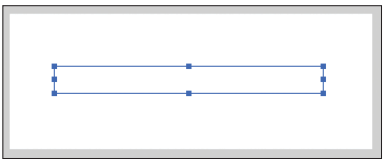
4.3 Creating A Flash Application Using Instance Method

In this unit we have discussed the major concepts related to instance methods. They are namely the bound and get and set methods. Now let us create an effective application based on the above concepts. The steps for creating the application are as follows:

- Open a blank document. If the document is opened in Flash ActionScript 2.0, it should be set here.
- Select the Text Tool and create a text box
- Set the text type as Dynamic Text, font as _sans and assign 'a' as

the variable name of the text box

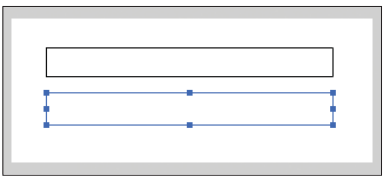
- Now click on the Text Tool again, and create another text box


Creating a Text Box

- Set the text type as Dynamic Text, font as _sans and assign 'b' as the variable name of the text box.


Setting the text box to dynamic text

- Now in Layer 1, right click in the first keyframe and click on the 'Actions' to open the Action Script window.


Creating another Text Box

- Here type this code:

```
var user:Login2 = new Login2("John");
var userNameStr:String = user.userName;
a=userNameStr
user.userName = "Della";
b=user.userName
```


Properties Window

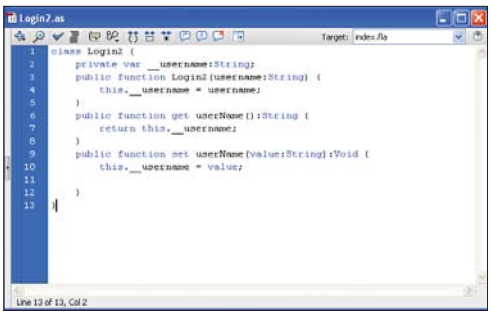

Actions Frame

Here, a new object, called 'Login 2' is created with the help of the new operator and the name 'John' is passed as parameter. The get method is then called. Next another variable named userNameStr is declared which is of String type. Next the set method is called. The string 'Della' is stored in the variable username which is further stored in the variable 'b'.

- Now open a new ActionScriopt file and type the following script as shown below:

```
class Login2 {
  private var __username:String;// a class that
contains a property called userName:
  public function Login2(username:String) {
        this.__username = username;
  }
  public function get userName():String {
        return  this.__username;//getUserName()
returns the current value of userName
  }
  public function set userName(value:String):Void
{
        this.__username = value;//sets the value
of userName to the string parameter passed to the
method.

  }
}
```

First a class 'Login2' is declared that contains a property called 'username'. This variable username is declared as String type. Next a constructor having the same class name i.e 'Login2' is defined and the parameter is passed to it. Next a get method is defined which returns the current value of the user name. Finally the set method is declared which sets the value of userName to the string parameter passed to the method.



Class Declaration

Functions

5.1 Basics of a Function

Functions represent one of the most important concepts that are used in any type of programming environment. They are basically blocks of code that are used for carrying out specific tasks and can be reused many times in the program. In the case of Flash ActionScripting, generally two types of functions are found. They are the methods and function closures. Depending on the context in which a function is defined, it can be called a function or a method closure. The function can be called a method only if it is defined as part of a class definition or it is attached to an instance of an object. It is called a function closure if it is declared in any other manner.

Calling Functions

A function may be called in many ways. One way of calling a function is by using its identifier followed by a parenthesis operator (). This parenthesis operator is mainly used to enclose function parameters that the programmer wants to send to the function. Consider the example given below. Here the trace() function, which is a top level function in the Flash Player API, is used.

```
trace("Use trace to help debug your script");
```

When a function without parameters is called, an empty pair of parenthesis is used. For instance the user can use the Math.random() method which does not take any parameters but generates random numbers. This is shown by an example given below.

```
var randomNum:Number = Math.random();
```

Defining Functions

In Flash ActionScripting, there are two ways in which a function can be defined. One way of defining a function is by using the

function statement and the other way is by using a function expression. The technique chosen by the programmer depends on the preference of programming style, which may be static or dynamic in nature.

Function Statements

Function statements are defined as the preferred techniques for defining various functions in strict mode. The various parts of a function statement are as follows:

- The function name
- The parameters, in a comma-delimited list enclosed in parentheses
- The function body - This is the ActionScript Code that is executed when the function is called and is enclosed in curly braces.

Let us consider an example where a piece of code creates a function that first defines the parameter, and then later invokes that particular function by using the word “hello” string as the parameter value.

```
function tr1(aParam:String)
{
     trace(aParam);
}

tr1(“hello”); // hello
```

Function Expressions

Another important way of declaring a function is to use an assignment statement with the help of a function expression — this is sometimes referred as a function literal or an anonymous function.

The various parts of an assignment statement containing a functional expression start with the var keyword which is followed by:

- The function name

- The colon operator (:)
- The Function class to indicate the data type
- The assignment operator (=)
- The function keyword
- The parameters, in a comma-delimited list enclosed in parentheses
- The function body–that is, the ActionScript code to be executed when the function is invoked, enclosed in curly braces

Consider the following code where a particular is declared using the function expression.

```
var tracePara:Function = function (aParam:String)
{
     trace(aParam);
};
tracePara("good morning"); // hello
```

It should be noticed that the function name need not be specified as in the case of a function statement. Besides this there is another major difference between function expressions and function statements — i.e a function expression is mainly an expression rather than a statement. This implies that any function expression does not have the capacity to stand on its own as is the case with a function statement. On the other hand a function expression can only be used as a part of a statement usually in the form of an assignment statement. This can be demonstrated by the below statement.

```
var tr1:Array = new Array();
tr1 = function (aParam:String)
{
     tr1(aParam);
};
tr1[0]("hello");
```

In Flash ActionScripting, the function statements are more readable and allow the programmer to override the final key-

words. The majority of function statements that are used create a stronger bond between the identifier — i.e the name of the function and the code which is used within the method body. However the value of the variable can be changed by using an assignment statement and at the same time the link or connection between a function expression and a variable can also be severed. To cope with this problem, the programmer can declare the variable as a constant, but this makes the code harder to read and prevents the use of the override and final keywords.

Return Values From Functions

A function can return a value by the use of the return statement, followed by that particular value or expression that the programmer wants to return. This can be demonstrated with the help of the following example:

```
function doubleNum1(baseNumber:int):int
{
     return (baseNumber * 2);
}
```

It is to be noted that a function is terminated by using a return statement. This ensures that all statements which lie below the return statement will not be executed. This can be shown by an example given below.

```
function doubleNum(baseNum1:int):int {
     return (baseNum1 * 2);
      trace("after return"); // This trace statement will not be executed.
}
```

In the case of a strict mode, the value that is returned by the function should match the specific return type that is selected by the programmer. This can be demonstrated with the help of the example shown below; this generates an error in strict mode as the function does not return a valid value.

5.2 Methods

The word 'method' is synonymous to function and belongs to a part of the class definition. When an instance of the class gets created, a method gets bound to that interface. A function can be conveniently declared outside a class but on the other hand a method cannot be used apart from the instance to which it is attached. In general a method is defined using the function keyword. A function statement can also be used as shown below.

```
public function sample1():String {} // here sample1 is declared as a function
```

Besides this a variable can also used to which a function expression can be assigned. This can be shown by the example given below.

```
public var sampleFun1:Function = function () {}
// here sampleFun1 is used as a variable to which a function expression can be assigned
```

5.3 Modular Functions In ActionScript

In Flash ActionScripting, the concept of modular functions is widely used for developing various applications. One such application can be the falling of snowflakes. The user can take the help of the reusable modular function for creating this application.

Let us take an application where modular function is used. Consider the code given below:

```
function update()
{

if (action == "run")
{
if (direction == "right")
this._x++;
else
```

```
this._x--;
}
else if (action == "jump")
{
this._y += this.yspeed;
this.yspeed++;
}
}
```

The code defined above is the generic code which uses a function called update() for updating an object. It is observed however that code written in this manner makes it difficult to add extra functions. To overcome this problem, the function can be made modular.

Making Code Modular

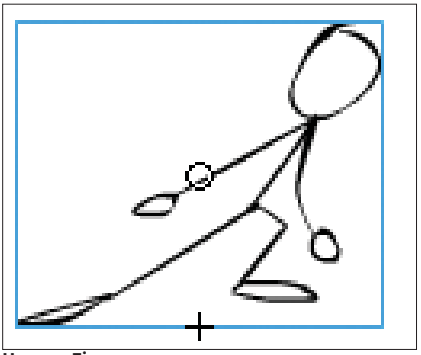
The above code can be modified by using a different type of function call which is shown below.

```
function update()
{
action = action();
}
```

5.4 Creating A Flash Application Using Function

In the above program segment, the code becomes much more simplified as all the conditional statements are omitted and only a function referrer is used for determining the next action. The parts of the code are broken into smaller segments for solving the problem with the generic function.

In this unit we have explained the various aspects of Functions, i.e. defining and calling functions, function statements and returning values from functions. Besides these, the concepts related to method formation and modular functions have also been discussed. Now let us create an effective application based on the above concepts. The detailed steps are given below.



Human Figure

- Draw a human like figure as shown below
- Select the figure and convert it into a movie clip
- Double click on the movie clip to edit the same
- Now from 5th frame onwards, insert key frames up to 12th frame
- From 5th to 12th frame add some movements to the hands and legs of the movie clip to make it a running man
- Now in the first frame we have a man standing and from 5th frame a man is running
- Now insert a new layer.
- Insert a keyframe in the 5th frame of this new layer
- Now insert a label in the first and key frame and set the label name of the first key frame to stand and the 5th key frame to run
- Now add a new layer and then add blank keyframes to the third, fourth and twelfth frames
- In the third keyframe add the following script

  gotoAndPlay (animation);

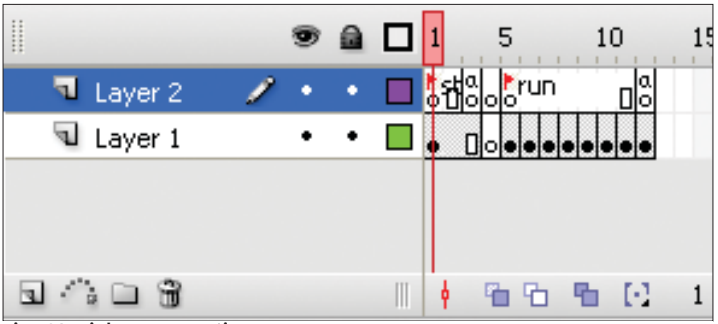


Fig 503: Linkage Properties

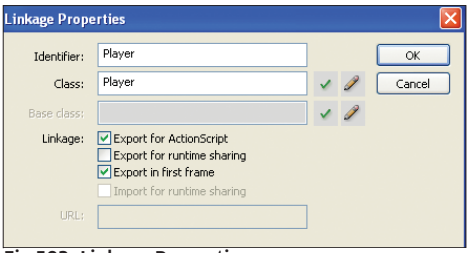


Fig 503: Linkage Properties

- In the fifth keyframe add the script gotoAndPlay(animation);
- Now come back to Scene1
- In the library palette, you can see the icon of the movie clip, right click on it and select the Rename option
- Rename the movie clip as player
- Again click on the movie clip, right click on it and select the Linkage option
- Here set the Identifier and the Class name as 'Player'
- Check two boxes i.e 'Export for ActionScript' and 'Export in first frame' under linkage
- Now right click on the movie clip situated on the stage.
- From the drop down list choose Actions.
- In the Action Script Panel add the following script

```
onClipEvent(enterFrame)
 {
  this.update();
}
```

- Now select the Text Tool and with it write the following text on the stage - "press left or right to run"
- Remember that the type of the text must be Static.
- Now open a new ActionScript file and type the following script as shown below

```
Script

class Player extends MovieClip
{
  private var action:Function;
  private var dir:Boolean;
  private var animation:String;
```

```
function Player()
{
        // init variables
        action = stand;
        animation = "stand";
        dir = true;
}
function update()
{
  action = action();

_xscale = (dir && _xscale < 0) || (!dir &&
_xscale > 0) ? -_xscale : _xscale;
}

function stand()
{
  if (animation != "stand")
        {
                animation = "stand";
                gotoAndPlay(animation);
        }

        if (Key.isDown(Key.RIGHT))
        {
          dir = true;
                return run;
        }
        else if (Key.isDown(Key.LEFT))
        {
          dir = false;

                return run;
        }

  return stand;
```

```
    }

        function run()
    {
                if (animation != "run")
        {
            animation = "run";
            gotoAndPlay(animation);
        }

        if (Key.isDown(Key.RIGHT))
        {
          dir = true;
            _x += 10;
            return run;
        }
        else if (Key.isDown(Key.LEFT))
        {
          dir = false;
            _x -= 10;
            return run;
        }

        // no key pressed, player stands
      return stand;
    }
}
```

Explanation of the Script:

```
class Player extends MovieClip
```

This is the first line of the program. Here a class named 'Player' is defined which is able to control the position of the 'MovieClip'. The keyword 'extends' is used so that all the functions and variables from the 'MovieClip' class can be inherited to the 'Player' class.

```
private var action:Function
private var dir:Boolean
private var animation:String
```

This part declares three types of variables which are named as action, dir and animation. The line private var action: Function references the current action function. The next line declares the variable dir as a Boolean type which relates to the right direction if true and to the left direction if false. The last line declares the variable animation as a String type and defines the frame label of the animation that should be played

```
function Player()
  {
        // init variables
        action = stand;
        animation = "stand";
        dir = true;
  }
```

Here the constructor named Player having the same name as that of the class is defined which initialises the variables. The member variables action (as a referrer to a function), animation (a String that contains the name of the animation that should be played) and dir are initialized in this part.

```
function update()
  {
    action = action();

        // set direction (mirror movieclip if
player faces to the left)
        _xscale = (dir && _xscale < 0) || (!dir
&& _xscale > 0) ? -_xscale : _xscale;}
```

This part of the code defines a modular function named update — i.e. instead of using many if-else statements a function

referrer is used to determine the next action. The next line of code within this function sets the direction of the mirror MovieClip. Here xscale is a variable which stores the value of the result of the conditional operation that uses the variable dir and xscale.

In the stand function, first it is checked whether the movie clip is moving or not. If the animation of the clip is in a running position then, "stand" is stored in animation. This value is then passed within the gotoAndPlay method. Next the Key.isDown method is used within the if-else if loop which determines the type of key that is pressed i.e left key or right key.

In the run function the motion of the movie clip is checked. The string "run" is stored within the animation which is passed within the gotoAndPlay method. Next the type of key (left or right) is checked. Depending on the nature of key — i.e. right or left key — the position of the player gets changed; i.e. the player moves to about 10 units per frame to the right or left. If neither of the keys is pressed, the player stands.

Arrays

6.1 What is an Array?

In a scripting language an array is a list that includes some objects that are maintained in a certain structural order. The properties of an array are identified by this order that indicates the position of those objects in the structure of the script.

To make it simpler, we can say that an Array is a container of some values or it is a structure of various data types. Here the contents, i.e. data, are more important than the container, i.e. the array.

Say, in a bookshelf we usually organize our books according to the subject or category. We usually keep apart science books from literature. So, each rack of this shelf includes books that are categorically different from the other types of books. At the same time they share a common quality, i.e., they are all books. If we compare these different types of book with different types of data, then we can compare the bookshelf to an array. Here the contents, i.e. books, are the most important thing to us. Similarly the data are also the most significant item in an array. The significance of the array (like the bookshelf) is that it can manage the data. Another important point with arrays is that we can form or structure the arrays as per our need.

The elements of an array possess various data types such as number, strings, dates, objects etc.

Here we have an example of an array of the first decade of this century:

```
var myArr:Array = new Array();
myArr[0] = "2000";
myArr[1] = "2001";
myArr[2] = "2002";
myArr[3] = "2003";
myArr[4] = "2004";
myArr[5] = "2005";
myArr[6] = "2006";
```

```
myArr[7] ="2007";
myArr[8] ="2008";
myArr[9] = "2009";
```

We can also write the above array of the first decade of 21st century in a short way, i.e. we can club the data type within a single brace. Now let us look at the following example:

```
var myArr:Array = new Array("2000", "2001", "2002", "2003", "2004', "2005", "2006", "2007", "2008", "2009");
```

At the same time an Array may include various types of data as in the following example:

```
["year", 2000, "month", 1,"date", 10]
```

We can use arrays in order to store various lists of objects. While loading data from some remote servers we may receive various types of data just like an array of nested objects. The norm is to use [0] as the first index number and then the index number follows according to the numeric order: 1, 2, 3, 4, etc.

In ActionScript variables are the most commonly used data container but a single variable can contain only one value, whereas the arrays can include more than one value and these data or the values can be customised separately.

Now before going through the details of arrays, we should look at the various important types of array as it will help us to develop a basic understanding of the various features of array:

Associative Array: The Associative Array can use string keys in order to recognize individual elements.

Example:

```
var monitorInfo:Array = new Array();
monitorInfo["set"] = "Flat Panel";
monitorInfo["declaration"] = "1600 x 1200";
trace(monitorInfo["set"], monitorInfo["declaration"]);
// output: Flat Panel 1600 x 1200
```

Key: The keys are the strings or the objects that we use in order to identify an element in the Associative Array as well as in a dictionary.

Dictionary: The Dictionary is a type of Array where the contents consist of key and value. Here the key is used in place of the numeric index in order to identify an element.

Indexed Array: an indexed array can store any element that is included in the number element. Here the individual elements are identified by using their number or index. These Indexed Arrays can use a 32-bit unsigned integer for the index number.

Here we should always remember that the maximum size of an indexed Array is $2^{32} - 1$ or 4,294,967,295. We will inevitably face a run time error if we attempt to create an Array that is larger than this maximum size. Now let us look at the example below:

Example:

```
// Use Array constructor.
var myArray:Array = new Array();
myArray.push("first");
myArray.push("second");
myArray.push("third");
trace(myArray); // output: first,second,third

// Use Array literal.
var myArray:Array = ["first", "second", "third"];
trace(myArray); // output: first,second,third
```

Element: elements are the items in an array. We will discuss elements in detail in the next section.

Multidimensional Array: a multidimensional array includes the items that are usually themselves identified as arrays and not as single values. This is an array of arrays. We will discuss multi-dimensional arrays in detail in the last section of this unit.

6.2 The Anatomy of an Array

An Array is composed of two things:

- Array elements
- Index number

The main content name of an array is called the array element. In the above example the years are the array elements. The unique numbers that are included in every line of code is called the index number of the array. These index numbers are extremely useful as we can avoid using different names for each array. Too many names of an array might lead to confusion. The index number always stars from 0. We can work with these index numbers in order to set or recover the value of the element. These elements' number or the index number can be inserted or deleted from any stage of the array as per our requirements. While working with ActionScript we will often find that in some positions of arrays some elements are absent. These gaps or absent arrays are called Sparse Arrays.

6.3 Creating Arrays

There are two ways to create an Array:

- By using Data Literal
- By using the Array() function

Let us discuss in detail:

- By using Data Literal

While creating arrays by using data literals we need to type the entire element that we want to insert in the arrays.

Now let us see how we can create an array by using array literals. Let us look at the syntax of the array literals;

[Expression1, Expression2, Expression3]

We can see here that this type of array always starts and ends with a square bracket. We can use any type of relevant expression. Let us go through some examples of this type of arrays. Here we should not forget to notice that each of these Arrays includes different types of data.

Example:

```
 [8, 9, 72];                          //    Simple
numeric elements
   ["Laswell", "Cromwell", "Ruskin"]          /   /
Simple string elements
   [2, 5, 3 + 6]                  // Numeric expres-
sions with operation
   [userName, alternativeName, "fare", "dark"]   //
Variables and strings as elements
   ["first days of the month", [1, 2, 3]]         //
A nested Array literal
```

For comparison, let's do the same thing with the Array( ) constructor:

```
   new Array(8, 9, 72)
   new Array("Laswell", "Cromwell", "Ruskin")
   new Array(2, 5, 3 + 6)
   new  Array(userName,  alternativeName,  "fare",
"dark")
   new  Array("first  days  of  the  month  ",  new
Array(1, 2, 3))
```

By Using The Array() Function

Now let us see how we can create an array using the Array() constructor. With the Array() constructor we insert the new operator first. Then we insert the Array keyword and a parenthesis.

There are three ways to create an array by using the Array constructor.

We will get an empty array if we call the Array constructor without any argument. Here the 'length' property of the array class can be used in order to verify whether the array includes any elements.

Here is an example where there is no argument in the array constructor:

```
   var name:Array = new Array();
   trace(name.length); // output: 0
```

If we use a number as a parameter of the array constructor,

then an array will be created that will maintain the same length where the value of the element is 'undefined'. Here we must set an unsigned integer as the argument and it must be between the values 0 and 4,294,967,295. Here we have an example where the code can call the Array constructor by using a single numeric argument:

```
var name:Array = new Array(3);
trace(name.length); // output: 3
trace(name[0]); // output: undefined
trace(name[1]); // output: undefined
trace(name[2]); // output: undefined
```

Lastly, we can create an Array if we call the constructor and a list of elements is passed as a parameter. Here, in these arrays the elements can correspond to the parameters. See this example where three arguments to the Array constructor can be passed:

```
var names:Array = new Array("Jack", "Jill",
"Jazz");
trace(names.length); // output: 3
trace(names[0]); // output: Jack
trace(names[1]); // output: Jill
trace(names[2]); // output: Jazz
```

6.4 Referencing Array Elements

We usually use number to reference elements but we can also name them. We can use the associative arrays or hashes for using named elements. We cannot operate named array elements by using array methods like push (), pop () etc. An array that includes one named element as well as two numbered elements will have a length of 2 and not 3.

The square brackets are used in order to add an element that we can get back later by using the name. We use strings in place of numbers in the square brackets.

Look at the following expression:

```
ArrayName[elementName] = expression
```

In the above expression the elementName indicates a string. Look at the example below:

```
var significantDates = new Array( );
significantDates["myBirthday"] = "Sept 20";
significantDates["sistersBirthday"] = "Sept 7";
```

Let us see how we can use the dot operator:

```
ArrayName.elementName = expression
```

In the above expression the elementName indicates nothing but an identifier, and it is not a string. Let us look at the example below:

```
var significantDates = new Array( );
significantDates.myBirthday = "Sept 20";
significantDates.sistersBirthday = "Sept 7";
```

Let us assume that we know an element's identifier (for example, myBirthday in the significantDates Array). Now let us see how we can access it.

There are two ways of accessing it and we can use any one of the two:

```
var goMarketing = significantDates["myBirthday"];
var goMarketing = significantDates.myBirthday;
```

While attributing a value to a named element when we use an element with square brackets, the elementName should always be an identifier when using dot operators. Here we do not use strings.

6.5 Determining the Size of an Array

The size of an array can be determined by the 'length' property of the array. The current number of the element can be specified by this 'length' property. In order to access the 'length' property of an Array the doc operator is used. See the expression below:

```
ArrayName.length
```

The 'length' property of an Array indicates the number of Array elements that are included in an Array. Look at the examples below:

```
myCatalog= [20, 25, 52];
```

```
   trace(myCataloglength);  // Displays: 3
   myRemark = ["fisrt option", "second option",
"third option"];
   trace(myRemark.length); // Displays: 3, the num-
ber of                          //  elements,  not
the number of words or characters
   frameLabels = new Array(24);      // Note the sin-
gle numeric //argument used with the Array( ) con-
structor
   trace(frameLabels.length);  // Displays: 24
```

The 'length' property of an array always maintains a certain rule, such as, it is always one greater than the index number of its last element. Say the indices of an array are 0,1,2,3,4. Suppose the index 4 has a length of 5. On the other hand an array includes elements where the indexes are 0,1,2,3,4 and here the index 60 includes a length of 61.

Another thing that we should always remember is that the last element of an array is always myArray[myArray.length - 1]. This is so as the index numbers start from 0 and not from 1.

While attempting any changes in an array the changes are also incorporated in the 'length' property so that the changes are reflected. These changes include adding as well as removing elements from an array. The 'length' property can be set in order to add or remove elements at the end of an array.

By using the 'length' property of an array, a loop can be created and here the loop can access all the elements of an array. In any programming language this kind of looping using the elements of an array is a very basic task.

Example:

Look at the example below, where we have given code to search an array:

```
   // Create an Array
   var soundtracks = ["mechanical", "hip hop",
"pop", "other", "unique"];
   // Check each element to see if it contains "hip
hop"
```

```
for (var i = 0; i < musictracks.length; i++) {
  trace("Let us test the element: " + i);
  if (soundtracks[i] == "hip hop") {
    trace("The place of 'hip hop' is index: " +
i);
    break;
  }
}
```

Example:

See the example below where we have given an example of a generalised array searching function:

```
function searchingArray (whatArray,
searchElement) {
  // Check each element to see if it contains
searchElement
  for (var i = 0; i < whatArray.length; i++) {
    if (whatArray[i] == searchElement) {
      return i;
    }
  }
  return null;
}
```

Let us see how we can use this search function in order to check whether 'Fritz' belongs to the usernames Array. Here, this username Array is an Array of sanctioned usernames.

```
if (searchArray (userNames, "Fritz") == null) {
  trace ("Unable to get the password");
} else {
  trace ("Welcome");
}
```

6.6 Adding And Removing Elements To An Array

We know that an array holds various elements and their related index numbers. We have already discussed that we can refer to an array using these index numbers. The elements of an array hold

data i.e. numbers, Booleans, strings, array objects. At the same time it is sometimes left empty. While debugging an application these index numbers are used. We have just discussed that the array 'length' property will return the value in addition to the index number. If the index number is 6 then the length will return 7. It indicates that 6 undefined values can be inserted into the arrays.

Here we have given an example that shows how we can create a new array and how we can delete an item from the required index. At the same time it shows the way we can add as well as replace data at the required index of an array:

```
var monthArr:Array = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
delete monthArr[6];
trace(monthArr); // Jan,Feb,Mar,Apr,May,Jun, undefined,Aug,Sep,Oct,Nov,Dec
trace(monthArr.length); // 12
monthArr[6] = "JUl";
trace(monthArr); //Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, //Oct, Nov, Dec
```

Though we have deleted the item from index 6, the array length does not change here and it remains 12. One thing that we should not forget here is that the array index does not disappear from here, rather it turns to a blank string.

6.7 Multidimensional Arrays

Up to now we have been discussing the single dimensional array. In a spreadsheet the single dimensional arrays include a single row or a single column. Now let us discuss multidimensional arrays. In order to create a multidimensional array we need to create arrays within arrays. We can define these as nested arrays. To put it simpler we can say that a multidimensional array is created when in an array the array elements are nothing but arrays themselves.

Two dimensional arrays are the most simple form of multidimensional arrays. Here we can imagine that the array elements are structured in both rows and columns. If the rows are the first dimension of the arrays then we can define the columns as the second dimension of the array. Let us look at the example below:

Say we are processing an array that includes three rows and two columns. This is a spreadsheet concerning products. Here, each row is used for each individual product. Among the two columns one is used for the quantity and the other is used for the price of the product. We can treat the elements as columns. Now look at the example below:

```
var row1 = [5, 3.99];   // Quantity 5, Price 3.99
var row2 = [3, 8.99];   // Quantity 3, Price 8.99
var row3 = [2, 60.99];  // Quantity 2, Price 60.99
```

Now let us place the rows into a container array that is used as a container and named as the spreadsheet:

```
var spreadsheet = [row1, row2, row3];
```

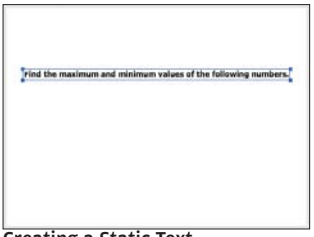
The total cost of the order can be searched if we multiply the amount and the cost of each row. We could also add all these together. The elements of a two dimensional Array can be accessed by using two indices where one index is used for the row and the other Index is used for the column. Here the first row's array of two columns can be represented by the expression spreadsheet[0]. We can use spreadsheet[0][1] in order to access the second column in the first row of the spreadsheet. Look at the following:

```
// Create a variable to store the total cost of
the order
var total;
// Now find the cost of the order. For each row,
multiply the columns
// together, and add that to the total.
for (var i = 0; k< spreadsheet.length; k++) {
  sum += spreadsheet[i][0] * spreadsheet[i][1];
}
trace(total);  // Displays: 120.99
```

Multidimensional arrays can behave just like other normal arrays. We may happily use all of the array methods to operate on the data that are stored in a multidimensional array.

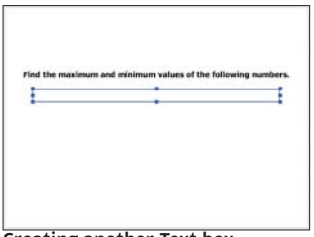
6.8 Creating an application using Arrays

In this unit we have learned the various aspects of arrays, i.e., array creation, referencing array elements, array size determination, adding elements etc. Now let us create a practical application based on the above concepts. The steps are defined below.



Creating a Static Text

- First open a blank document. If the application is opened in flash ActionScript 2.0, then this should be set here.
- Now by selecting the Text Tool Type "Find the maximum and minimum values of the following numbers"
- Set the text type to Static text
- Now by using the text tool, draw a text box below it

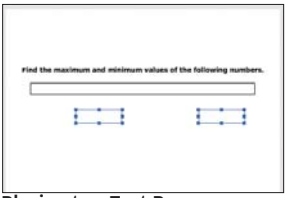


Creating another Text box

- Set the text type of this text box as Dynamic text
- Set the Alignment of this text box to Center
- Set the variable name of this dynamic text box to 'text'


Properties Window

- Create another two text boxes below this with the help of the Text Tool
- Set the text type of these text boxes as Dynamic text
- Set the Alignment of this text boxes to Center
- Set the variable names of these dynamic text boxes to 'mx' and 'mn'
- Create another two text boxes with the Text Tool and type "Max:" and "Min:"



Placing two Text Boxes


Code Window

- Set the text type of these text boxes to Static text
- Place these two static text boxes besides the above dynamic text boxes accordingly
- Now create another Layer and name it as Actions
- Select the first keyframe and right click on it
- Choose Actions
- The Action Script window opens
- Here type the following script

```
Script:
maximumValue = function (array) {
max = array[0];
for (i=0; i<array.length; i++) {
if (array[i]>max) {
max = array[i];
}
}
return max;
};
minimumValue = function (array) {
min = array[0];
for (i=0; i<array.length; i++) {
if (array[i]<min) {
min = array[i];
}
}
return min;
};
k = new Array();
k = [12, 5, 63, 7, 1, -1, 352, 36, 754, 75, 56,
74];
text=k;
mx=maximumValue(k);
mn=minimumValue(k);
```

- Once the script has been written, press (Ctrl+Enter) to run the application.

Explanation Of The Script

Before understanding the code, we should first consider a generic approach to the problem. The code deals with an array with a long list of numbers and our aim is to find the maximum and minimum values from that list.

The detailed form of the array is mentioned above. It is to be noted that the first number within the array always starts from the 0th element and the number of elements is always denoted by n-1.

```
maximumValue = function (array)
max = array[0];
```

In this line of code a new function known as the 'maximumValue' is declared that accepts an argument called array. Next a variable called mx is declared with an initial value which is the first value of the array. It is to be assumed that the first value with the array is a temporary maximum value. For instance if the initial value of an array is zero and the largest number within the array is a negative number, then the initial value declared within the array shall be considered as the maximum value.

```
for (i=0; i<array.length; i++)
```

The next part of the code deals with the for loop. Here a simple loop is initiated that counts from the value zero and finally reaches the end of our array. The statement i=0 initializes the variable i. The next part of the loop i<array.length makes the counter progressively pass through each value in the array until the end is reached.

```
if (array[i]>max) {
max = array[i];
}
```

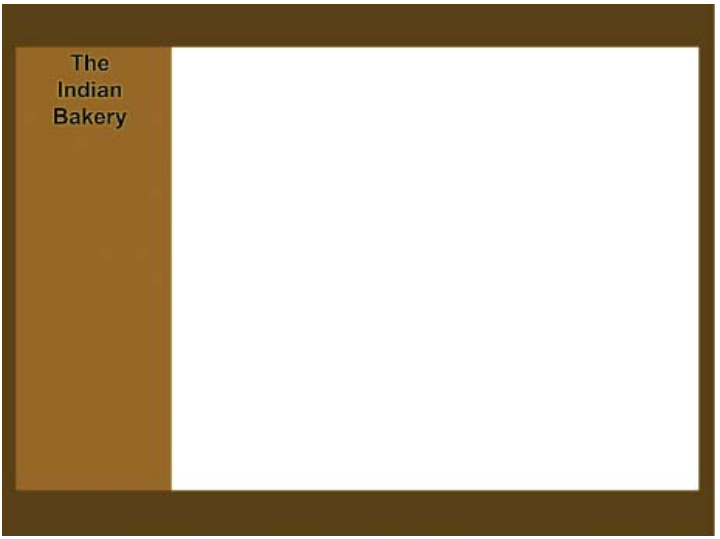
In the above lines of code, the number within the array is checked at the ith position. If the number in this position is greater than the maximum value, the current value is then set as the maximum value. The next statement returns mx as the maximum value of the array which is stored in mx.

Projects

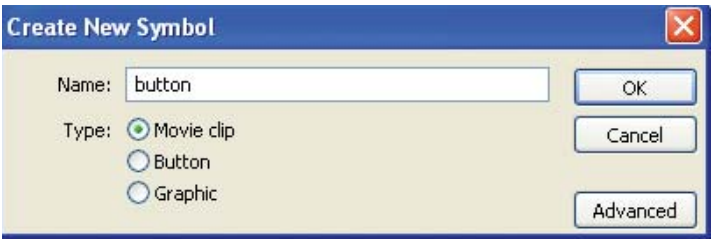
a) Project1 - A Complete Project based on XML parsing

Steps needed to make the application

- Open a blank document. If the document is opened in Action Script 2.0 then this should be set here.
- Now create an interface as shown below. This is the mother file. Here we have named it as **index.fla**
- First create a movie clip, name it as 'button'. This movie clip is used as a button

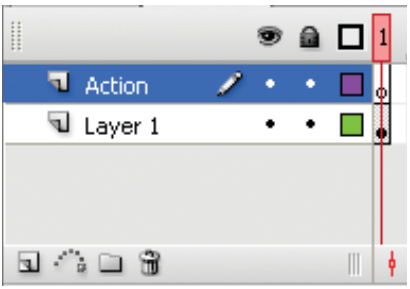


The Interface



'New Symbol' Dialog Box

- Put the movie clip inside the library, and right click on it.
- Now click on its 'Linkage' properties
- Set the name of the Identifier of the movie clip as 'button_nav'
- This movie clip (used as a button) is used as a menu item and is associated with a definite xml file.

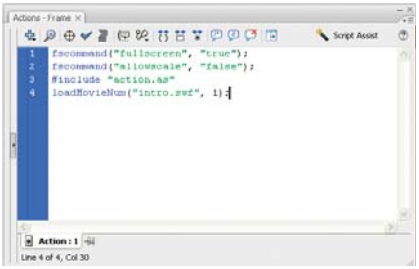


Creation of a New Layer

- Create a new layer and name it as "Action"
- Right click on the first key frame and choose 'Actions'.
- The Script window opens. Here add the following code. (Explanation given below)

```
fscommand("fullscreen", "true");
fscommand("allowscale", "false");
#include "action.as"
loadMovieNum("intro.swf", 1);
```

- For each menu item create a separate fla file e.g. **page1.fla** where the text along with the image shall be loaded as shown below.
- Here two buttons are created as **'Back'** and **'Next'** (for navigation purpose) which are common to rest of the fla files. The buttons are named as **'Symbol1'** and **'Symbol2'.**
- Create a new layer and name it as **'Action'**
- Right click on the first keyframe and choose **'Actions'.**



Actions Frame Window

- The Script window opens. Here add the following code. (Explanation given below)

```
#include "h-text-image.as"
#include "h-text-img1.as"
```

- In this way create four separate fla files i.e **page2, page 3, page4** and **page 5** as shown above.
- For each of these fla files, create a new layer and name it as an action

Creating Buttons

The 'Back' Button

- Right click on the first keyframe and choose 'Actions'.
- The Script window opens. Add the relevant code which ise as follows:

For Intro.fla

```
#include "h-text-image.as"
#include "h-text-img.as"
```

For Page2.fla

```
#include "h-text-image.as"
#include "h-text-img2.as"
```

For Page3.fla

```
#include "h-text-image.as"
#include "h-text-img3.as"
```

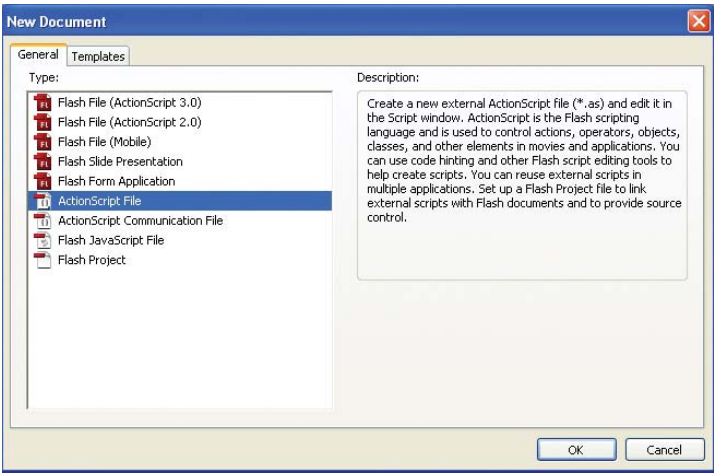
For Page4.fla

```
#include "h-text-image.as"
#include "h-text-img4.as"
```

For Page5.fla

```
#include "h-text-image.as"
#include "h-text-img5.as"
```

- Now open an ActionScript file and type the following script as follows:



Selecting ActionScript File

```
var myxml:XML = new XML();
myxml.ignoreWhite = true;
myxml.load("menu.xml");
myxml.onLoad = function() {
  _global.totalmenu =
this.firstChild.childNodes.length;
   _global.chpaterMc =
_root.createEmptyMovieClip("chapter", 1);
  _root.chapter._x = 25;
  _root.chapter._y = 170;
```

```
    for (i=0; i!=totalmenu; i++) {
            tName =
this.firstChild.childNodes[i].attributes.tName;
            mName =
this.firstChild.childNodes[i].attributes.mName;

            id =
this.firstChild.childNodes[i].attributes.id;
            button = _root.chapter.attachMovie("but-
ton_nav", "button_nav"+i, i);
            button.txt.text = tName;
            button._y = 25*i;
            button.xx = mName;
            button.onRelease = function() {
                  loadMovieNum(this.xx,1);
            };
            button.onRollOver = function() {
                  mycol = new Color(this.hover);
                  mycol.setRGB("0xececd7");
                  arrows = new Color(this.arow);
                  arrowcol.setRGB("0xcccccc");
                  txtcol = new Color(this.txt);
                  txtcol.setRGB("0xececd7");
            };
            button.onRollOut = function() {
                  mycol.setRGB("0xf4f5df");
                  arrowcol.setRGB("0xffffff");
                  txtcol.setRGB("0xffffff");
            };
    };
  };
```

- Save this file as **'action.as'** (Explanation given below)
- Now open another ActionScript file and type the following script as follows.
- Name this file as **'h-text-image.as'**

```
   style_fmt1 = new TextFormat();
   style_fmt1.align = "justify";
   this.createTextField("bodytext",
this.getNextHighestDepth(), 227, 81, 520, 422);
   bodytext.setNewTextFormat(style_fmt1);
```

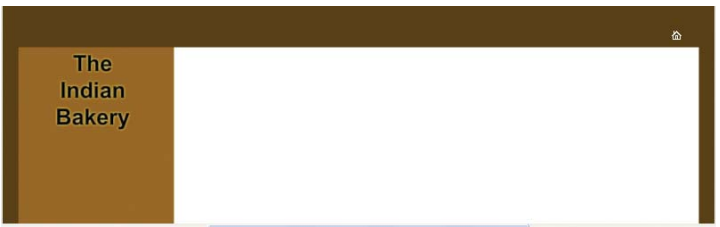
- Open another ActionScript file and type the following script. Save the file as **'h-text-img.as'** file.

```
   bodytext.html = true;
   bodytext.wordWrap = true;
   bodytext.multiline = true;
   bodytext.condenseWhite = true;
   bodytext.selectable = false;
   var myCSS = new TextField.StyleSheet();
   var cssURL = "style.css";
   myCSS.load(cssURL);
   myCSS.onLoad = function(success) {
     if (success) {
            bodytext.styleSheet = myCSS;
            bodytext.text = "CSS file loaded suc-
cessfully. You may proceed.";
     } else {
     bodytext.text = "CSS file failed to load!";
       }
   }
   myContent = new XML();
   myContent.ignoreWhite = true;
   myContent.load("intro.xml");
   myContent.onLoad = function(success) {
     if (success) {
            bodytext.text = myContent;
     } else {
            bodytext.text += "<br /> XML file
failed to load!";
     }
   }
```

- Now make another five new ActionScript files, named – **'h-text-img1.as', 'h-text-img2.as', 'h-text-img3.as', 'h-text-img4.as' and 'h-text-img5.as**' accordingly.

Note:
It is to be noted that for the various .fla files, i.e., intro.fla, page1.fla, page2.fla, page3.fla, page 3.fla, page 4.fla and page 5.fla, two action script files are called. One is ***h-text-image.as*** *file which is common to all the fla files and other is the* ***h-text-img.as*** *files related to that particular fla.*

- Here you can see that these sections of code are the as same as the 'h-text-img.as' file, except for one variation. Only the targeted XML files are different here.
- After making all the pages, again we have to open the 'index.fla' for some modifications.
- Now we will insert the 'Home' and 'Exit (X)' buttons.
- First we will insert the 'Home' button. So we have to insert the image of a home. Click on File.
- Then click on Import and then Import to Stage.
- Select the image and place it on top-right corner of the fla file as the image below.

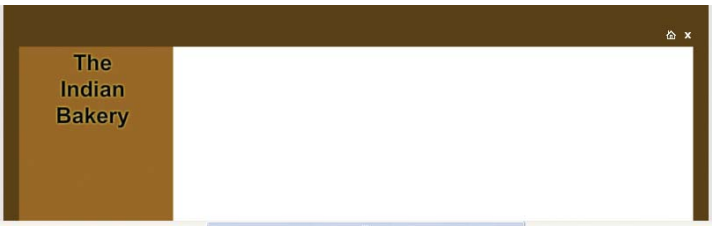


Placing the 'Home' Button

- Convert it into a button and name it as 'home-but'.
- Now right click on it and choose 'Actions'.
- In the Action Script window add the following code:

```
on (release) {
  loadMovieNum("intro.swf",1);
}
```

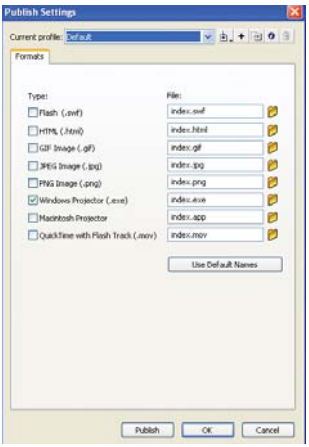
- Next we have to insert the 'Exit' button.
- So make a cross sign just beside the 'Home' button as the image below.



Placing the 'Exit' Button

- Convert it into a button and name it as 'exit'.
- Now type the following code for this button:

```
on (release) {
  fscommand("quit");
}
```



'Publish Settings' Window

- Now go to File and then Publish Settings.
- Publish the file as **'Windows Projector'.**
- Now we have to navigate all the pages, i.e. 'page1', 'page2' etc.
- So first we have to open the 'page1.fla' file.
- Here the 'BACK' button must be inactive, so select the 'NEXT' button and add this script in it:

```
on (release) {
loadMovieNum("page2.swf",1);
}
```

- Now export it as a swf file.
- Similarly, we have to add the following code snippets to the

respective files and export them as swf files.

For 'page2.fla' :-
'BACK' button:

```
on (release) {
  loadMovieNum("page1.swf",1);
}
```

The above code helps the user to navigate to the previous swf file, i.e., page1.swf.

'NEXT' button:

```
on (release) {
  loadMovieNum("page3.swf",1);
}
```

The above code helps the user to navigate to the next swf file, i.e., page3.swf.

For 'page3.fla' :-
'BACK' button:

```
on (release) {
  loadMovieNum("page2.swf",1);
}
```

'NEXT' button:

```
on (release) {
  loadMovieNum("page4.swf",1);
}
```

For 'page4.fla' :-
'BACK' button:

```
on (release) {
  loadMovieNum("page3.swf",1);
}
'NEXT' button:
on (release) {
```

```
    loadMovieNum("page5.swf",1);
  }
```

For 'page5.fla' :-
'BACK' button:

```
on (release) {
  loadMovieNum("page4.swf",1);
}
```

'NEXT' button:
Here the 'NEXT' button is inactive.

- You can see that the lines of code are almost the same, only the target swf files are different.
- Now in the same folder, we have to place the XML files and the CSS file.
- Also the images used must be kept in the same directory, within the 'images' folder.

Script and explanation of the major fla and .as files

This is the mother file which contains all the menu items. Clicking on each menu item opens up a new file.

Script

```
fscommand("fullscreen", "true");
fscommand("allowscale", "false");
#include "action.as"
loadMovieNum("intro.swf", 1);
```

The first line defines a method named as fscommand where the parameters "full screen" and "true" are passed. This method indicates that the application should only be run on a full screen. The next line **fscommand ("allowscale", "false")** passes two parameters "allow scale" and sets its to "false".

Next the preprocessor **#include** calls the actionscript file 'action.as'. Lastly the method **loadMovieNum** loads the intro.swf file.

Intro.fla
This is the next fla file that should be created after creating the Intro.fla.

Script

```
#include "h-text-image.as"
#include "h-text-img.as"
```

The script associated with this fla file includes two action script files which are h-text-image.as and h-text-img.as

Page1.fla
This is the file that is related to the first menu item of the application.

Script

```
#include "h-text-image.as"
#include "h-text-img1.as"
```

The script associated with this fla file includes two action script files which are h-text-image.as and h-text-img1.as

Page2.fla
This is the file that is related to the second menu item of the application.

Script

```
#include "h-text-image.as"
#include "h-text-img2.as"
```

The script associated with this fla file includes two action script files which are h-text-image.as and h-text-img2.as

Page3.fla
This is the file that is related to the third menu item of the application.

Script

```
#include "h-text-image.as"
#include "h-text-img3.as"
```

The script associated with this fla file includes two action script files which are h-text-image.as and h-text-img3.as

Page 4.fla

This is the file that is related to the fourth menu item of the application.

Script

```
#include "h-text-image.as"
#include "h-text-img4.as"
```

The script associated with this fla file includes two action script files which are h-text-image.as and h-text-img4.as

Page5.fla

This is the file that is related to the fifth menu item of the application.

Script

```
#include "h-text-image.as"
#include "h-text-img5.as"
```

The script associated with this fla file includes two action script files which are h-text-image.as and h-text-img1.as

Explanation of action.as

Script and its explanation

```
var myxml:XML = new XML();
```

The first line of the code creates an xml object named myxml. This is done by using a new method.

```
myxml.ignoreWhite = true;
```

This line of code defines that the xml object, i.e., myxml, that is created should ignore white spaces in the xml code.

```
myxml.load("menu.xml");
```

This line of code loads the file menu.xml inside the flash application using the load method.

menu.xml

```
<?xml version="1.0"?>
<menu >
  <item tName="Fruit Biscuits"
mName="page1.swf"/>
  <item tName="Cashew nut Biscuit"
mName="page2.swf"/>
  <item tName="Badam Pista Biscuit"
mName="page3.swf"/>
  <item tName="Chocolate Cashew nut"
mName="page4.swf"/>
  <item tName="Chocolate Bounty"
mName="page5.swf"/>
</menu>
```

Explanation:

The various item names are added within this xml document. Next the related swf files associated along with each item in the menu is defined and stored in the mName variable.

```
myxml.onLoad = function() {
          _global.totalmenu =
this.firstChild.childNodes.length;
```

Here, first the object myxml accesses the onload function with the (.) operator. The global variable "total menu" that is declared carries the total number of childnodes. Here it is 5 as five menu items are used in the application.

```
_global.chpaterMc =
_root.createEmptyMovieClip("chapter", 1);
```

This line of code creates an empty movie clip called 'chapter' in

level for loading the data from the xml file. This is done with the help of the method known as createEmptyMovieClip method.

```
_root.chapter._x = 25;
root.chapter._y = 170;
```

The above two lines of code fix the x and y position of the first menu related to the empty movie clip that is created.

```
for (i=0; i!=totalmenu; i++) {
          tName =
this.firstChild.childNodes[i].attributes.tName;
          mName =
this.firstChild.childNodes[i].attributes.mName;

          id =
this.firstChild.childNodes[i].attributes.id;
```

The above lines of code executes a for loop. Here the function receives the five values of tName,mName and id attributes from the xml file when i=0, 1, 2....

```
button = _root.chapter.attachMovie("button_nav",
"button_nav"+i, i);
```

Here "button_nav" is the linkage identifier of the movieclip "button" in the "index.fla" file. It is also attached with the empty movieclip "chapter". "Button" is a variable which contains the value of "chapter".

button.txt.text = tName;

Here "button" is associated with the dynamic text field 'txt' which stores the value of 'tName'.

```
button._y = 25*i;
button.xx = mName;
```

The above lines of code define the y position of the buttons which is increased with the value of i. The height of each button is fixed at 25px.

```
button.onRelease = function() {
loadMovieNum(this.xx,1);
```

Here the method loadMovieNum is used for loading the corresponding movie method mentioned in the xml.

```
button.onRollOver = function() {
mycol = new Color(this.hover);
mycol.setRGB("0xececd7");
arrowcol = new Color(this.arow);
arrowcol.setRGB("0xcccccc");
txtcol = new Color(this.txt);
txtcol.setRGB("0xececd7");
};
```

The above lines of code are related to the various settings of the button on roll over. The colour of the button and the arrow colour are set here.

```
button.onRollOut = function() {
mycol.setRGB("0xf4f5df");
arrowcol.setRGB("0xffffff");
txtcol.setRGB("0xffffff");
```

H-text-image.as file

This action script file is associated with the intro.fla file. This script file mainly deals with the formatting of the text file and creation of a empty text field called body text.

```
style_fmt1 = new TextFormat();
style_fmt1.align = "justify";
```

The above code creates a new style called fmt1. The alignment of this new style is then set to justify.

```
   this.createTextField("bodytext",
this.getNextHighestDepth(), 227, 81, 520, 422);
   bodytext.setNewTextFormat(style_fmt1);
```

The above code creates an empty text field called "bodytext" with the help of the createTextField method. The new text field that is created assumes the new style fmt1 that is created above. The getNextHighestDepth method sets the depth of the body text.

H-text-img.as file

This action script file is associated with the intro.fla file. The various settings associated with the 'body text' (created in h-text-image.as file) text box is declared within this file

```
   bodytext.html = true;
   bodytext.wordWrap = true;
   bodytext.multiline = true;
   bodytext.condenseWhite = true;
   bodytext.selectable = false;
```

The first line indicates that the textbox 'body text' should be html compatible and is set to true. The other properties associated with this text box are also fixed , i.e., word-Wrap is set to true, multiline is set to true (i.e. the text box can accept multiple lines of text), white spaces are removed by setting condenseWhite property to true. Only the selectable property is set to false.

```
   var myCSS = new TextField.StyleSheet();
```

Here a new style sheet object is added/ created by the new operator. The name of the new style sheet is given as myCSS.

```
   var cssURL = "style.css";
   myCSS.load(cssURL);
```

The above code specifies the location of the existing CSS file and then loads the current CSS file. The file style.css is described below

Style.css

```
h1{
font-family:Helvetica, Arial, sans-serif;
font-size:22px;
color:#5B3E00;
}

byline{
  font-family:Tahoma;
  font-size:13px;
  color:#000000;
  display:block;
  }
```

Here the various font parameters associated with the header h1 i.e font-family, font size and color are declared. These font parameters associated with the h1 header of the xml documents appear according to the above specifications.

The byline is the body tag and the various parameters associated with it i.e., colour, display, font-size and font-family are declared above. The text within this tag appears according to the above specifications.

```
myCSS.onLoad = function(success) {
  if (success) {
          bodytext.styleSheet = myCSS;
          bodytext.text = "CSS file loaded suc-
cessfully. You may proceed.";
   } else {
          bodytext.text = "CSS file failed to
load!";
     }
}
```

This line of code defines the onLoad handler, i.e. if the loading is successful the style sheet is loaded without any error. A message is generated on successful loading. On the other hand, if the loading is unsuccessful a message appears which indicates unsuccessful loading.

```
myContent = new XML();
myContent.ignoreWhite = true;
myContent.load("intro.xml");
myContent.onLoad = function(success) {
  if (success) {
        bodytext.text = myContent;//the value
of myContent is loaded into bodytext.
  } else {
        bodytext.text += "<br /> XML file
failed to load!"; } }
```

The first line indicates about the parsing of the 'myContent' variable with 'intro.xml' file. The ignoreWhite property is set to True. Then the 'intro.xml file gets loaded with 'myContent'. Now the loading is checked by a conditional statement, i.e., if the loading is successful, the xml content is seen within the text field, i.e., body text. Otherwise if the loading is unsuccessful, an error message related to unsuccessful loading is generated.

Intro.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<body>
<h1>Welcome</h1><br /><byline>Lorem ipsum dolor
sit amet, consectetuer adipiscing elit. Vivamus
suscipit aliquet nunc. Aliquam interdum turpis
vitae risus. Nullam dolor. Pellentesque at lectus
et pede egestas viverra. Donec aliquam tellus et
magna. Morbi pharetra nisl in nibh. Fusce iaculis.
Sed condimentum tellus nec diam. Maecenas id
metus.</byline><br/>
```

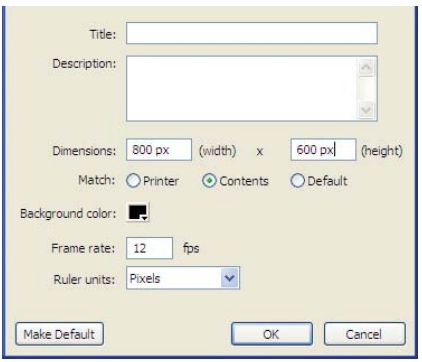
```
<img src="images/img.jpg">
</body
```

Explanation of Intro.xml

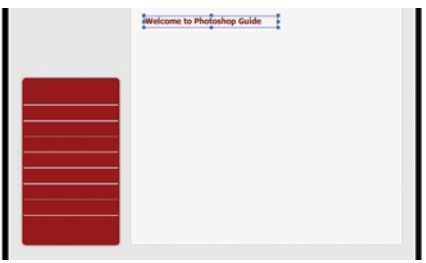
This file is very important as it contains the body part and the source of the image associated with the h-text-img file. The textual part written under the byline tag is written according to the specification mentioned in style.css file. Lastly the image source of the jpeg picture file associated with this .as file is defined with this .xml file

b) Project 2 - A Complete Project on CD Authoring

Developing a training CD on Photoshop



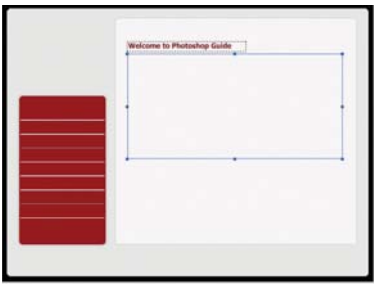
Document properties



Creating a Dynamic Text

- First open a blank flash document. If the document is opened in ActionScript 2 this should be set here)
- Set the size of the document to 800 x 600 pixels. This is shown in the figure below.
- Now click on the text box tool and create a text area
- Set the text area as 'Dynamic Text'
- Now type 'Welcome to Photoshop Guide' within this text area
- Select the text tool again and create another bigger text area
- Set the text type as 'Dynamic Text'

- Now create a text called "Introducing Photoshop" and convert it into a button
- This button is to be used as a menu item
- In this way, create six more buttons which are named as 'Workspace', 'Image size and resolution,' 'Making Selections', 'Layers and Blend Modes', 'Painting Tools' and 'Masking Layers'
- Each of these buttons used as menu items are related to a particular .fla file.
- Now we have to add action script for each and every button to make it navigable and interactive.



Creating a Bigger Text Area

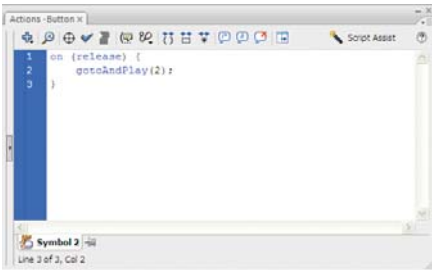
Text to Button

Creating More Buttons

- First let us take the button named 'Introducing Photoshop'. Right click on the button and select the option 'Actions' from the dropdown menu.
- Add the following script:

```
on (release) {
  gotoAndPlay(2);
}
```

- The above code takes the control to action that shall be performed in keyframe 2



Adding Script

- Next we shall consider then next button i.e. Workspace. Right click on the button

and again select the 'Actions' option. Here type the following script:

```
on (release) {
  gotoAndPlay(3);
}
```

- The above code controls the action that shall be performed in keyframe 3
- Next we shall consider the next button, i.e. image size and resolution. Right click on the button and again select the 'Actions' option. Here type the following script

```
on (release) {
  gotoAndPlay(4);
}
```

- The above code controls the action that shall be performed in keyframe 4
- Next we shall consider the next button i.e. 'Making Selection'. Right click on the button and again select the 'Actions' option. Here type the following script

```
on (release) {
  gotoAndPlay(5);
}
```

- The above code controls the action that shall be performed in keyframe 5
- Next we shall consider then next button, i.e., 'Layers and Blend Modes'. Right click on the button and again select the 'Actions' option. Here type the following script

```
on (release) {
  gotoAndPlay(6);
}
```

- The above code controls the to action that will be performed in

keyframe 6

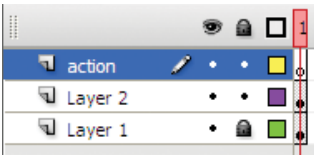
- Next we shall consider the next button, i.e. 'Painting Tools'. Right click on the button and again select the 'Actions' option. Here type the following script

```
on (release) {
  gotoAndPlay(7);
}
```

- The above code controls the action that shall be performed in keyframe 7
- Next we shall consider then next button, i.e., 'Masking Layers'. Right click on the button and again select the 'Actions' option. Here type the following script

```
on (release) {
  gotoAndPlay(8);
}
```

- The above code controls the action that shall be performed in keyframe 8
- Now take a new layer and name it as 'action'

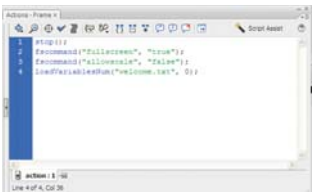


Creating an Action Layer

- In this layer, click on the first keyframe and right click on it. Select the actions option from the dropdown menu and type the following script:

```
stop();
fscommand("fullscreen", "true");
fscommand("allowscale", "false");
loadVariablesNum("welcome.txt", 0);
```

- Here the fscommand accepts two parameters, two parameters which are fullscreen and allows-


Adding Script

cale. The full screen is set to true while the allow scale is set to false. Next the method loadVariablesNum loads the txt document named welcome.txt.

- Insert seven more keyframes in this layer.
- Next select the second keyframe and type the following script:

```
stop();
loadVariablesNum("intro.txt", 0);
unloadMovie(image_mc);
unloadMovieNum(3);
```

- The above code first loads the text file named 'intro.txt'. The code also unloads the empty movie clip known as image_mc with the help of the unloadMovie method.
- Now click the third keyframe and type the following script :

```
stop();
unloadMovieNum(3);
this.createEmptyMovieClip("image_mc", 2);
image_mc._x=340;
image_mc._y=120;
loadMovie("images\\interface.jpg", image_mc);
```

- In the above code, the unloadMovieNum method unloads the movie clip from layer 3. Next an empty movie clip is created in layer 3 by the createEmptyMovieClip method. The x and the y coordinates of this movie clip are then set. Lastly the loadMovie method loads the interface.jpg image
- Now click the fourth keyframe and type the following script :

```
stop();
unloadMovie(image_mc);
loadMovieNum("page3.swf",3);
```

- The above code first unloads the empty movie clip, image_mc and then loads the swf file named page3.swf in the third layer

- Now click the fifth keyframe and type the following script :

```
stop();
unloadMovie(image_mc);
loadMovieNum("page4.swf",3);
```

- The above code first unloads the empty movie clip, image_mc and then loads the .swf file named page4.swf in the third layer
- Now click the sixth keyframe and type the following script :

```
stop();
unloadMovie(image_mc);
loadMovieNum("page5.swf",3)
```

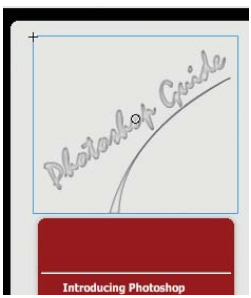
- The above code first unloads the empty movie clip, image_mc and then loads the swf file named page5.swf in the third layer
- Now click the seventh keyframe and type the following script:

```
stop();
unloadMovie(image_mc);
loadMovieNum("page6.swf",3);
```

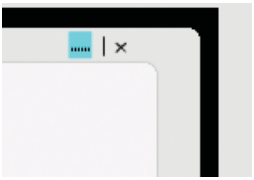
- The above code first unloads the empty movie clip, image_mc and then loads the swf file named page6.swf in the third layer
- Now click the eight key frame and type the following script:

```
stop();
unloadMovie(image_mc);
loadMovieNum("page7.swf",3);
```

- The above code first unloads the empty movie clip, image_mc and then loads the swf file named page7.swf in the third layer
- After this, create a movie clip with a label named 'Photoshop Guide' and place it above the menu items.



Adding Script



Sound Button

- Next create one button for starting and stopping the sound (which is an instance of anioff). Within the sound button two movie clips are inserted i.e anioff and anion.
- A new .fla is created in which the sound file (sound4.mp3) is loaded.
- Take a new layer.
- Now click on the first keyframe, and type the script which is related to the functioning of the sound button:

```
loadMovieNum("sound.swf", 50);
anioff._visible = false;
soundon._visible = false;
//song_sound.position = 0;

soundon.onRelease = function() {
  loadMovieNum("sound.swf",50);
  //song_sound.start(0, 700);
  anion._visible = true;
  anioff._visible = false;
  soundoff._visible = true;
  soundon._visible = false;
};
soundoff.onRelease = function() {
  unloadMovieNum(50);
  //song_sound.stop();
  anion._visible = false;
  anioff._visible = true;
  soundoff._visible = false;
  soundon._visible = true;
};
```

- The above code first loads the sound swf file. Initially both the objects anion and 'sound on' are set to false. Next the code defines the function when the sound button is pressed, i.e., the sound is started. First the sound file is loaded, the anion is set to

true and anioff is set to false. The visibility of soundon is then set to false while the visibility of sound off is set to true.

- The reverse happens when the sound button is pressed i.e sound-off is pressed and the sound stops.
- Now create another button for closing the application. Place this quit button at the top right corner of the application.
- Right click on the button and type the code by clicking on 'Actions'

```
on (release) {
fscommand("quit");
        }
```

- Next create two buttons named 'back' and 'next' which are created for navigation purposes.
- Click on the 'back button'. Right click on it and click on the 'Actions' options
- Now type the following code

Back and Next Button

```
  on (release) {
    prevFrame(); // the user is taken to the pre-
vious frame
  }
```

Similarly click on the 'next' button. Right click on it and click on the 'Actions' options.Now type the following code

```
  on (release) {
    nextFrame(); // the user is taken to the next
frame
  }
```

- While creating the page 5.fla and page 6.fla files select the first key frame and right click on it. Now click on the Actions option

and type the following code:

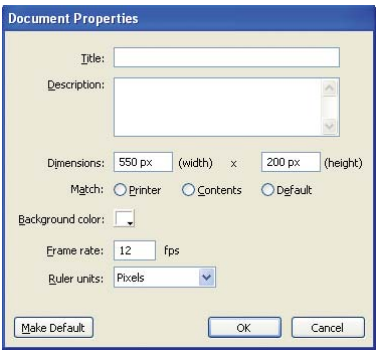
```
loadVariablesNum("layer.txt", 3); // in case of
page5.fla. This code loads the layer.txt file

loadVariablesNum("paint.txt", 3); // in case of
page6.fla. This code loads the paint.txt file .
```

Follow similar steps to develop a complete project.

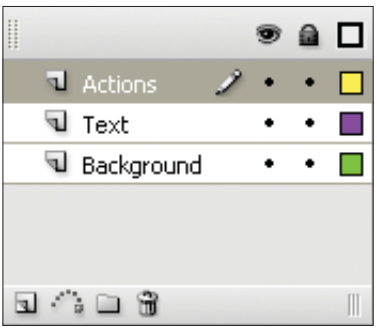
Application Resources

8.1 Digital Clock

While making the application, keep the dimensions as 550 x 200 px, colour: white, frame rate: 12 fps

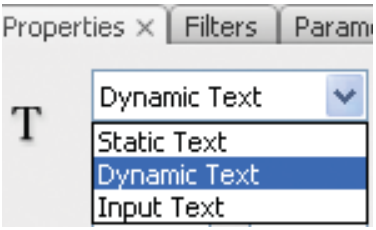


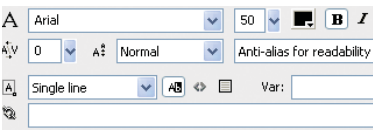
- Now let us learn to create a realistic digital clock in a Flash application.
- A blank document can be seen on the screen.
- In the Timeline window, insert three layers.
- Name the first layer as "Background", second layer as "Text" and the layer as "Actions" respectively.

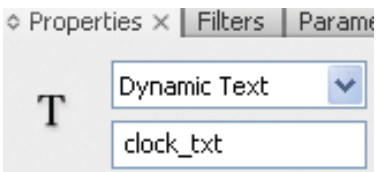


- Next, select the first frame of the Background layer.
- Here, we make a suitable framework for the digital clock.
- Now, select frame 2 and then press `[F5]`.
- Next, go to the Frame 1 of the Text layer.

- Here, insert Dynamic text field in the work area.
- For inserting Dynamic text field, select the Text Tool from the Tool box.
- In the Properties Inspector tab, select the Text type as Dynamic from the drop down menu as shown in the figure below.
- Set the Font as Arial, Size as 50 and Color as Black.
- Now, click on a portion of the work area.
- The digital clock will appear in that portion.
- Click on the Selection Tool from the Tool box.
- Next, select frame 2 and then press [F5].
- Name the Dynamic text field as "clock_txt" in the Instance name.
- Next, go to frame 1 of the Actions layer.
- Add this script in the Actions panel.

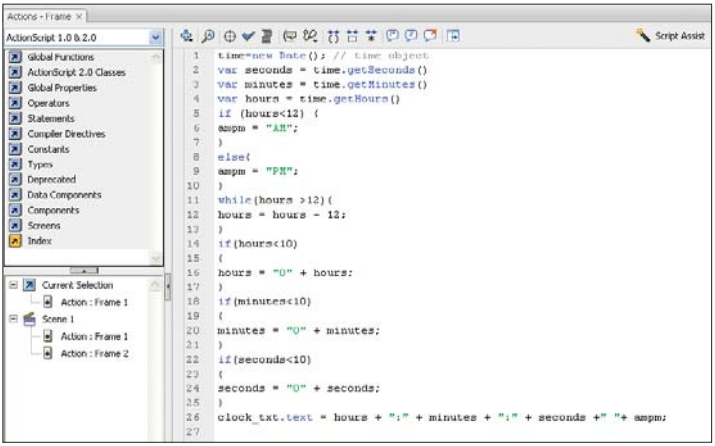




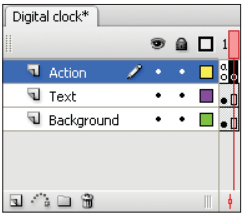



```
time=new Date();
var seconds = time.getSeconds() //this is the
variable for seconds
var minutes = time.getMinutes()//this is the vari-
able for minutes
var hours = time.getHours()//this is the variable
for hours
if (hours<12) {
ampm = "AM";
}
```

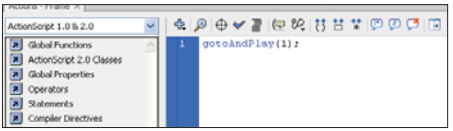
```
else{

ampm = "PM";
}
while(hours >12){

              hours = hours - 12;
}
if(hours<10){
             hours = "0" + hours;
}
if(minutes<10){
            minutes = "0" + minutes;
}
if(seconds<10){
            seconds = "0" + seconds;
}
clock_txt.text = hours + ":" + minutes + ":" +
seconds +" "+ ampm; // this will combine the
strings of hour, minute, second and AM/PM
```



- In the Actions layer, select frame 2.
- Now insert a new Keyframe in this frame by pressing F6.



- Keep the play head on frame 2 and go to the action panel.

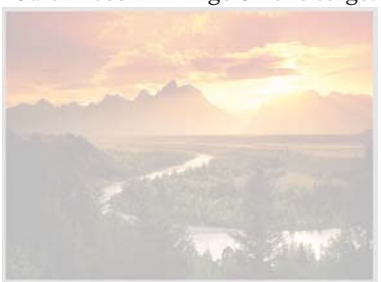


- Here, copy and paste the script.

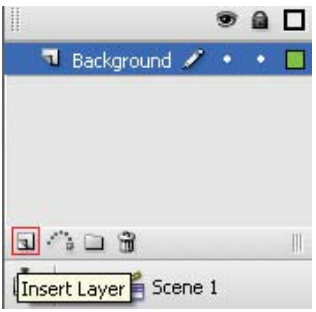
```
gotoAndPlay(1);
```

8.2 Drag Mask

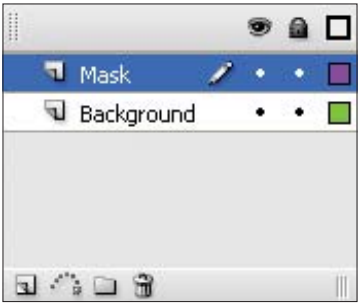
You can see an image on the stage.



- Now, on clicking on the Insert Layer icon, insert a new layer

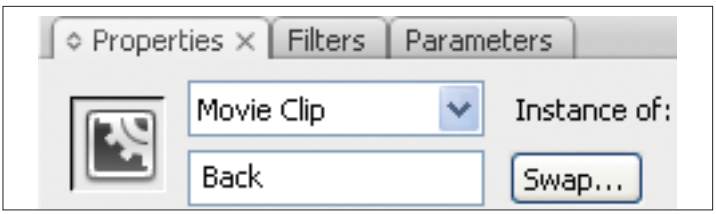


- Drag the image from the library to stage.
- Select the image, press [F8] and convert the image to a symbol
- Now select the Movie clip option, name the Movie clip as 'Draggable mask'
- Click OK.



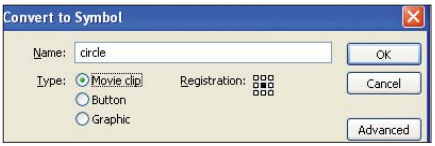
- Now double click on the image to enter inside the Movie clip
- Add a new layer and name it 'Mask'
- Notice your image is in the Background layer.
- Select the Background layer inside the Movie clip and press [F8].

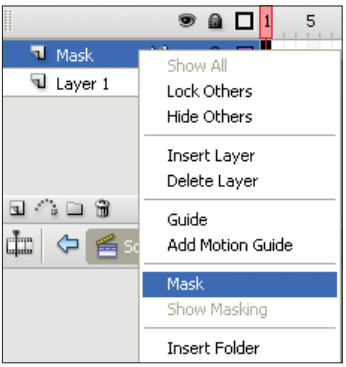
- Select Movie clip and name it as 'Background'.



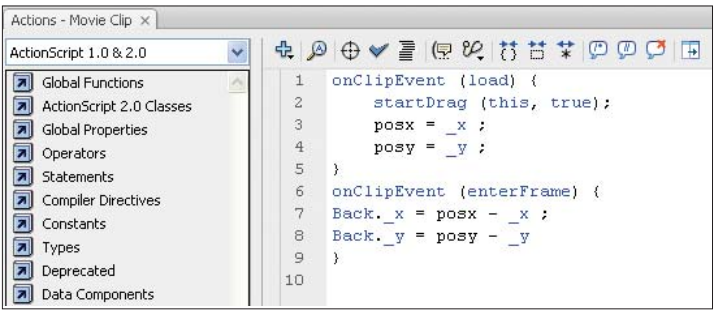
- Set the instance name of the Movie clip as 'Back'.
- Create a circle in the first frame of the mask layer.
- Select the circle and press [F8].

- Convert it to Movie clip and name 'circle'.
- Right click over the Mask layer and choose mask.





- You will see that the image has been masked within the circle.



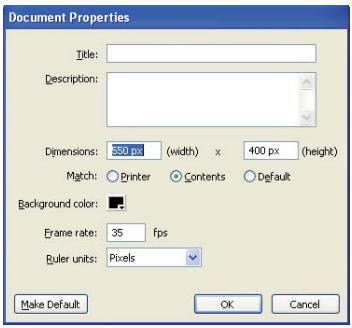
- Now return to scene 1.
- Right click on the Movie clip.
- Choose 'Actions', the Action panel opens.
- Here add this code:-

```
onClipEvent (load) {
    startDrag (this, true);
    posx = _x ;
```

```
    posy = _y ;
}
onClipEvent (enterFrame) {
Back._x = posx - _x ;
Back._y = posy - _y
}
```

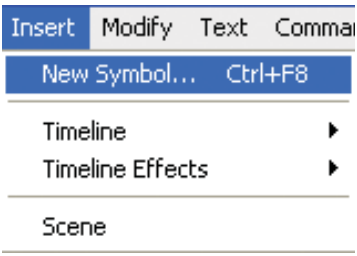
8.3 Falling Snow

While making the application, keep the dimensions as 550 x 400 px. Set then background colour to black and frame rate to 35 fps.



- Let us learn to create an animated falling snow Effect.
- You can see an image on your screen.

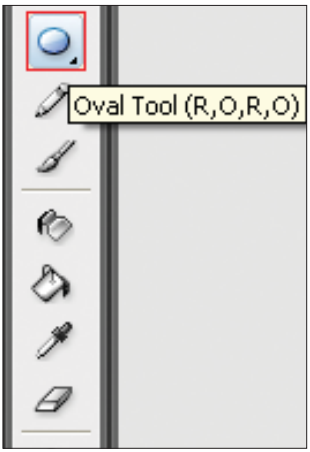
- Let us now start to generate snow.
- Apply Insert > New Symbol.
- Now in the dialog box click on 'Advanced' button.

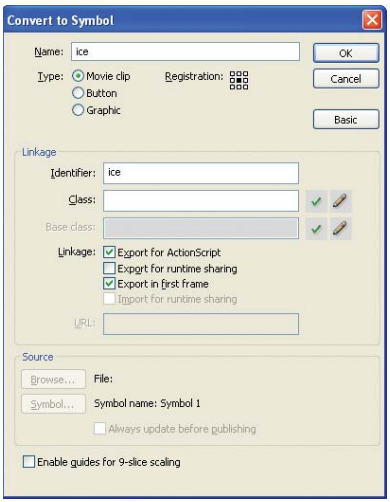


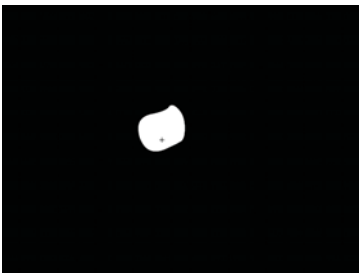
- Once we have added the new symbol, we will name it as 'ice'.
- Now change its property as the followings:
  - Name: ice
  - Type: Movie clip
  - Identifier: ice

- Linkage: Export for ActionScript – checked on
- Export in first frame – checked on.

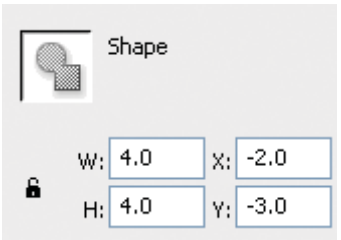
- Select Oval Tool from the Tool box.





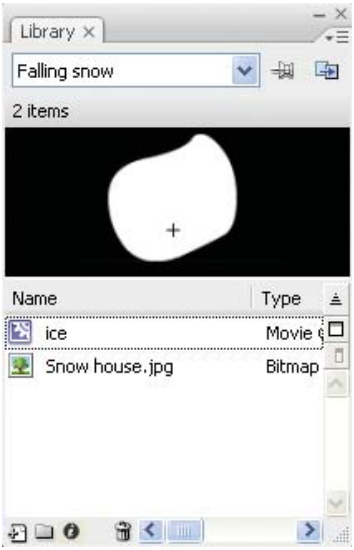


- Now set the Stroke colour to 'No Color'.
- Set the Fill colour to White.
- Draw a small circle.
- Click on the Selection Tool.
- Select the object.

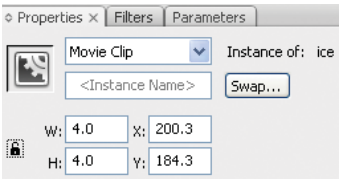


- Now set the properties as follows:
  - W: 4.0
  - H: 4.0
  - X: -2.0
  - Y: -3.0

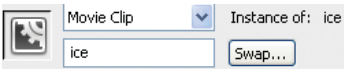
- Now select 'Scene 1'.
- Press [Ctrl] + [L] to open the Library panel.



- From the library now drag the movie clip 'ice'
- Next modify its properties as:
  - W: 4.0
  - H: 4.0
  - X: 200.3
  - Y: 184.3



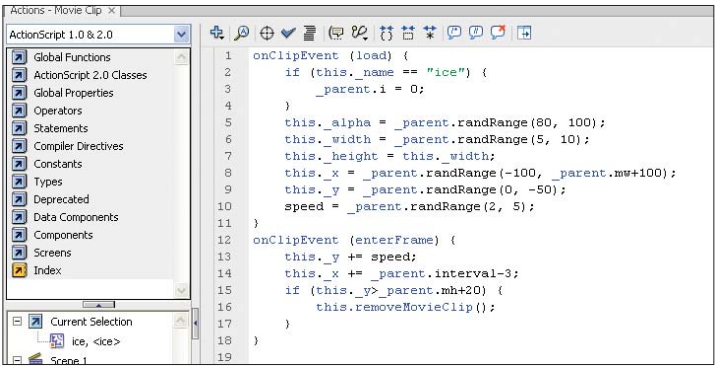
- Give it an instance name as 'ice'.



- Now open the Actions Script Panel.
- Now add the following code.

```
onClipEvent (load) {
    if (this._name == "ice") {
          _parent.i = 0;
    }
    this._alpha = _parent.randRange(80, 100); //
this will define the opacity

    this._width = _parent.randRange(5, 10); //this
will define the dimension
    this._height = this._width; //this will define
the dimension
    this._x = _parent.randRange(-100,
_parent.mw+100);
    this._y = _parent.randRange(0, -50);
    speed = _parent.randRange(2, 5); //this
```

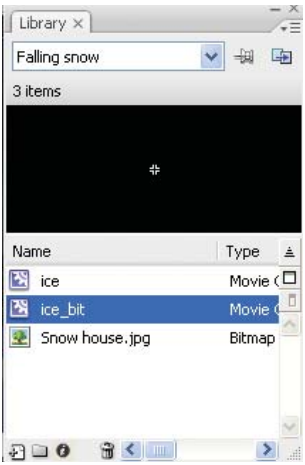
```
defines the speed of the movieclip

}
onClipEvent (enterFrame) {
     this._y += speed;
     this._x += _parent.interval-3;
     if (this._y>_parent.mh+20) {
            this.removeMovieClip();
     }
}
```



- Again create a Movie clip with a name 'ice_bit'.
- To do this, again go to Insert > New Symbol.
- Type 'ice_bit'
- Click OK.

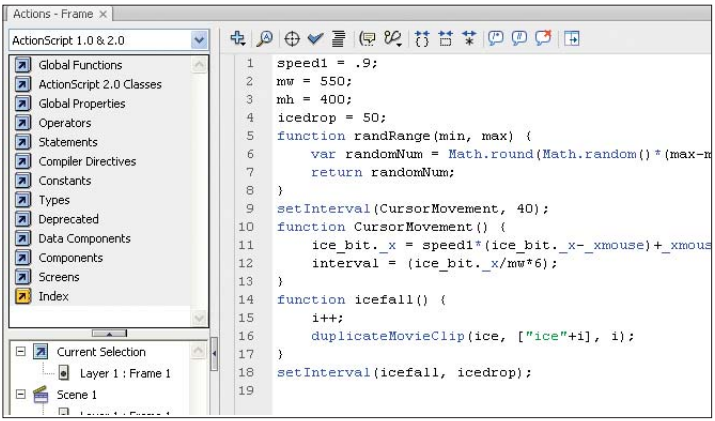
- Keep it empty and go back to Scene1.

- Drag the new movie clip from the Library panel.



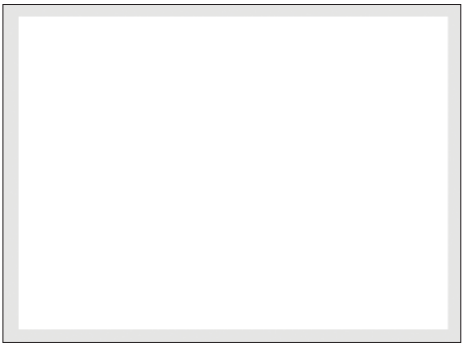
- Now give it an instance name as ice_bit.
- Select first frame of 'Scene 1' and add these code to Actions panel.

```
speed1 = .9;
mw = 550;
mh = 400;
icedrop = 50;
function randRange(min,
max) {
  var randomNum =
Math.round(Math.random()*(max-min))+min;
     return randomNum;
}
setInterval(CursorMovement, 40);
function CursorMovement() {
     ice_bit._x = speed1*(ice_bit._x-
_xmouse)+_xmouse;
     interval = (ice_bit._x/mw*6);
}
function icefall() {
     i++;
     duplicateMovieClip(ice, ["ice"+i], i);  //
this helps to increase the number of snow flakes
}
setInterval(icefall, icedrop);
```



8.4 Fireworks

- Open a blank document. You can see a blank stage on the screen. This is shown below.



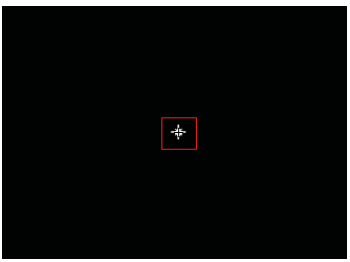
- Set the Background colour of the stage to black.

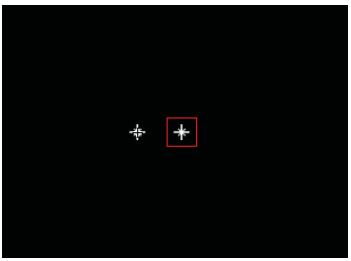
- Now create a star type sign.
- Convert this start shaped type sign into a Movie clip.
- For doing this, select the star sign and press [F8].
- A dialog box appears.
- Name it as “Star_mc” and select Movie clip.
- Click OK.

- Now double click on the star symbol to enter inside the movie clip.
- Now select the star again and then press [F8].
- Again the same dialogue box appears.
- Now in this case give it a name as "Star1_mc" and convert it to movie clip.
- Click OK.

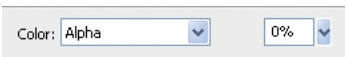
- Double-click on it to enter inside the 2nd movie clip.
- Here select the star and press [F8].

- Now convert it to Graphics.
- Name it as "Star_gr".
- Click on OK.

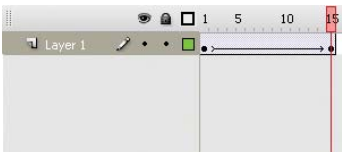




- Insert a Keyframe in frame 15 and move the star a bit.

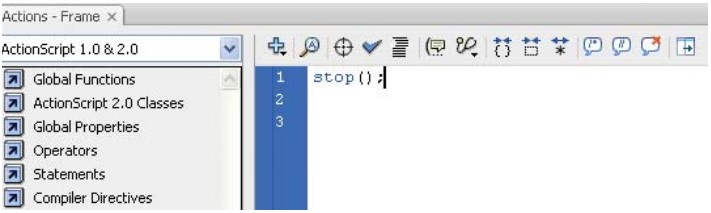


- Also change the Alpha value of the star to 0 in the 15th frame.



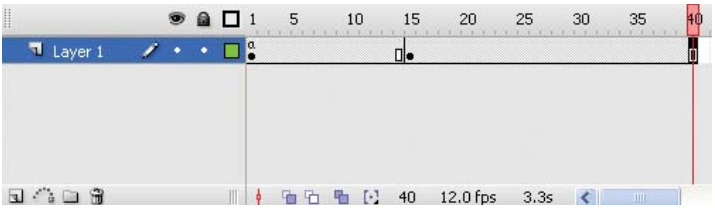
- Now create a motion Tween between frames 1 and 15.
- Now select the last Key Frame and then right click on your mouse and choose Actions.
- Add this script.

```
stop();
```



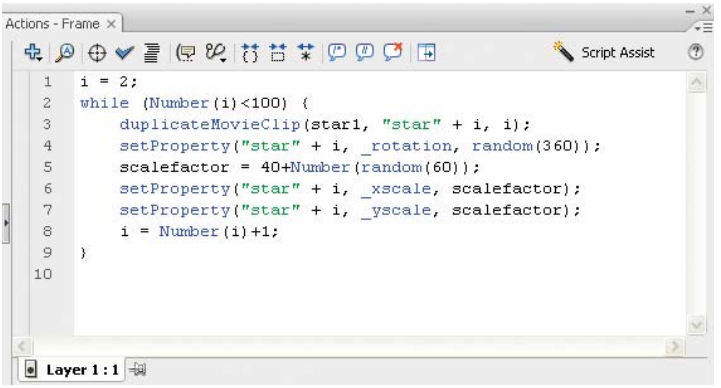
- Now after the above action is completed, come back to the Star_mc movie clip.
- Give the movie clip an instance name.
- Here the movie clip is named as 'star1'.

- Add a Keyframe in the 15th frame.
- Click on 40th frame and press [F5] to insert frame.



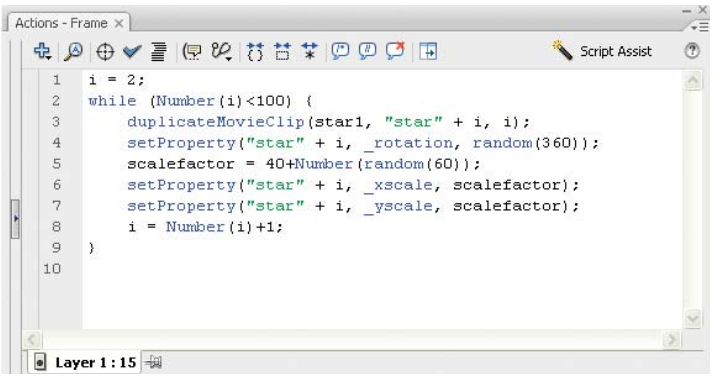
- Now select Keyframe 1 and add this script.

```
i = 2;
while (Number(i)<100) {
     duplicateMovieClip(star1, "star" + i, i);
     setProperty("star" + i, _rotation, ran-
dom(360));
     scalefactor = 40+Number(random(60));
     setProperty("star" + i, _xscale, scalefactor);
     setProperty("star" + i, _yscale, scalefactor);
     i = Number(i)+1;
}
```

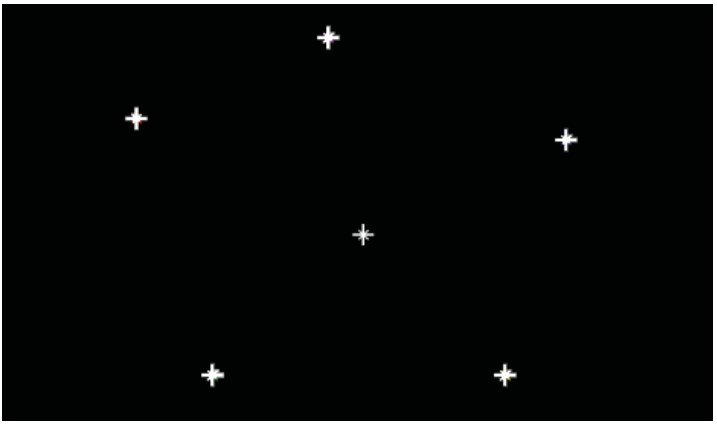


Now select Keyframe 15 and add this script.

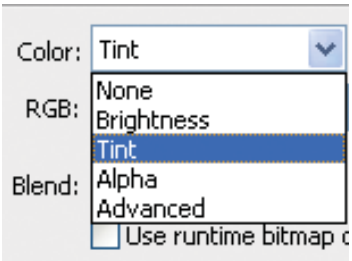
```
i = 2;
while (Number(i)<100) {
     duplicateMovieClip(star1, "star" + i, i);
     setProperty("star" + i, _rotation, ran-
dom(360));
     scalefactor = 40+Number(random(60));
     setProperty("star" + i, _xscale, scalefactor);
     setProperty("star" + i, _yscale, scalefactor);
     i = Number(i)+1;
}
```



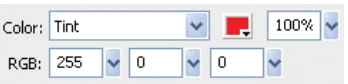
- Now come back to Scene 1.
- After the above action, copy the movie clip and paste it several times.



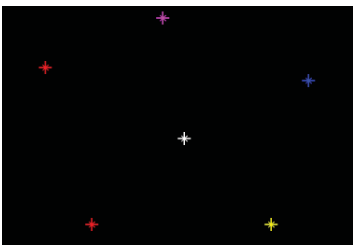
- Now select the symbol and go to Properties Inspector.



- Select Tint from the Color option.
- Now a particular color is to selected for Tint.

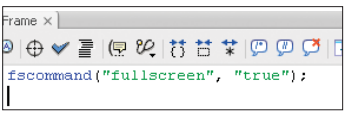


- Repeat these steps for the other symbols.
- Select frame 1 and add this script.



```
fscommand("fullscreen", "true");
```

- Now the animation is complete.

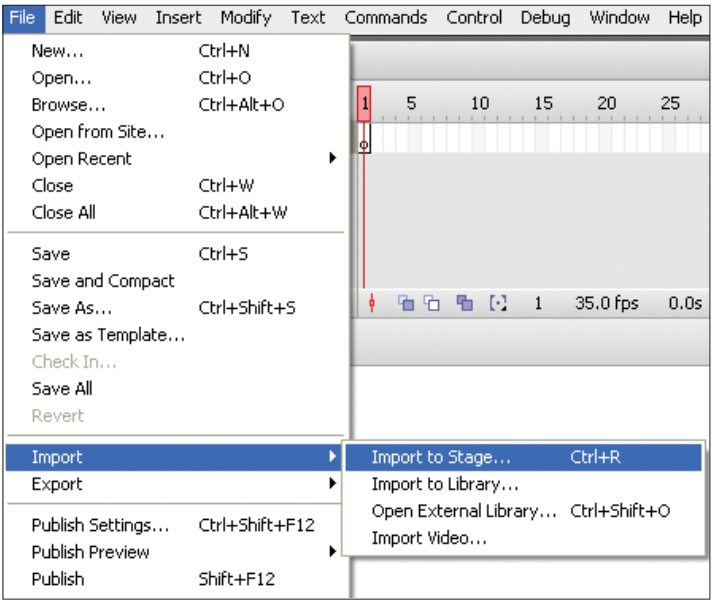


8.5 Flying Hearts

Flying Hearts is an excellent animation that can be created in Flash. Let's learn how to create flying hearts animation in Flash. This is a simple animation and is fun to create. So let's begin.

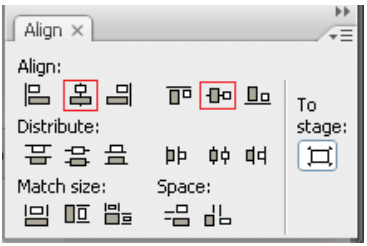
- You can see a blank Flash document on the screen.
- Double click on a particular layer
- Rename the layer as "Background"

- Now import an image.
- So, click on File > Import > Import to Stage.

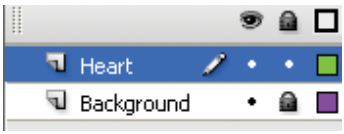


- Press [Ctrl] + [K] to open the Align Panel.
- Click on the Align to stage icon.

- Now click on Align horizontal center icon.
- Next click on Align Vertical center icon.

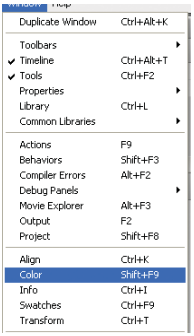


- Now lock the layer.

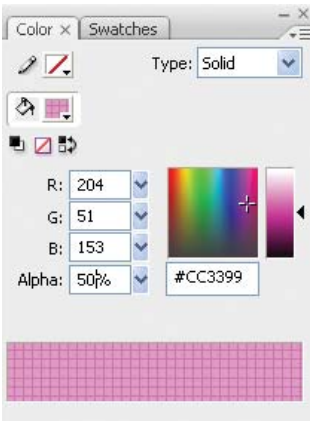


- Insert a new layer.
- Rename this layer as “Heart”.

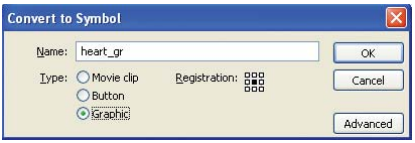
- Now draw a small heart.
- Now change the colour of the heart.
- So, click on Window > Color.
- In the Color Panel, set the Stroke colour as None.
- Select the Fill colour as pink. (#CC3399)

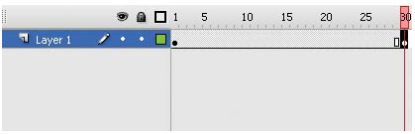


- Set the Alpha value to 50%.
- Keep it selected and press `[F8]` to convert it into a Movie clip.
- Click on OK.



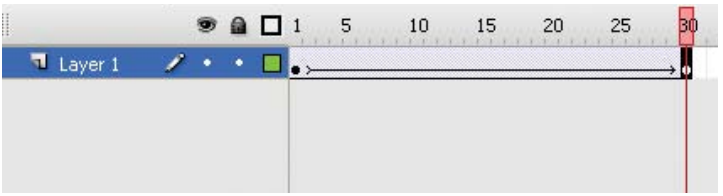
- Now, double click on the Movie clip to edit it.
- Again press `[F8]` to convert the heart to a Graphic symbol.





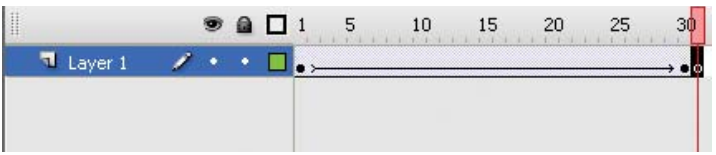
- Next, click on frame 30 and insert a keyframe.
- Now, increase the size of the heart on the last frame and move its position.
- Now, create a Motion Tween between frames 1 and 30.

- Now, we will create the animation.
- To do this, select the next frame to the last frame.
- Press `[F7]` to insert a blank keyframe here.



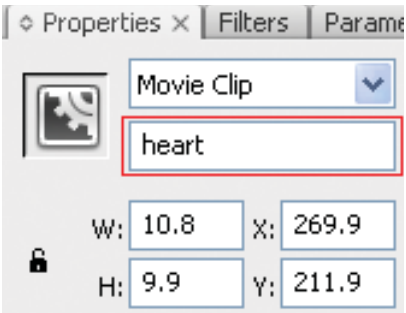
- Now, select it and open the Actions Script Panel.
- Add this script.

```
stop();
```



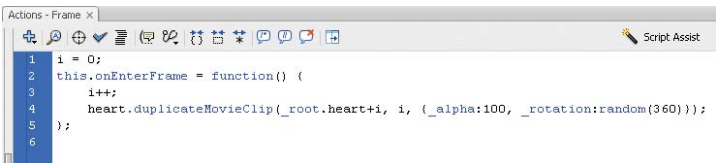
- Now, move back to the main Timeline.
- Select the heart.
- In the Instance name type name as “heart”.

- Select the first frame from the TimeLine
- Now, open the Actions Script Panel.
- Add this script:



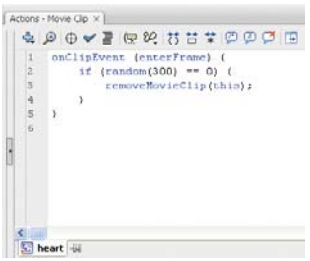

```
i = 0;
this.onEnterFrame =
function() {
  i++;
```

```
heart.duplicateMovieClip(_root.heart+i, i,
{_alpha:100, _rotation:random(360)});
```




```
};
```

- Now click on the heart movie clip.
- Again open the Actions Script Panel.
- Add this script.

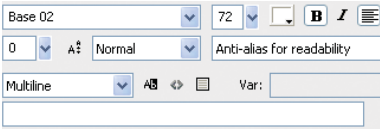


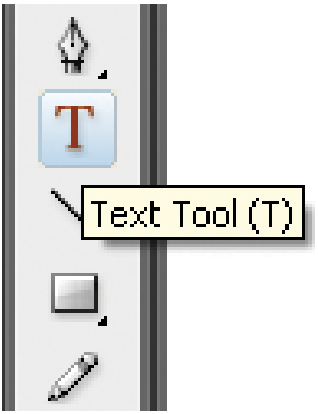

```
onClipEvent (enterFrame) {
    if (random(300) == 0) {
    removeMovieClip(this);
    }
```

}

8.6 Mask Text Effect

- First select the image on your screen
- Now insert a new layer.
- Rename the new layer as





"Text".

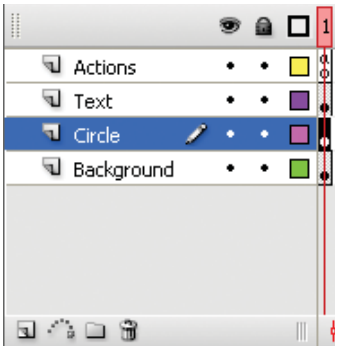
- Go to the Tool box and select the Text Tool.
- Set the Font as Base 02 and size as 72.
- After you have done it type the text.

- Here we type 'NATURE'.
- Now click on the Selection Tool.
- Again insert a new layer.
- Rename this layer as "Actions".

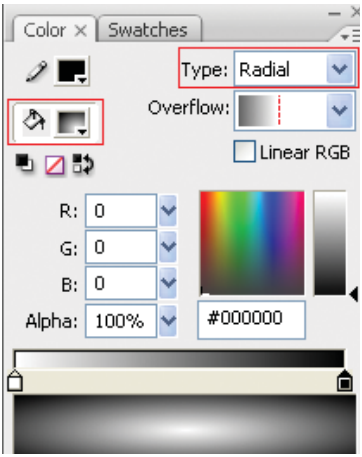
- Now open the Actions Script Panel and add this code:

```
startDrag
```

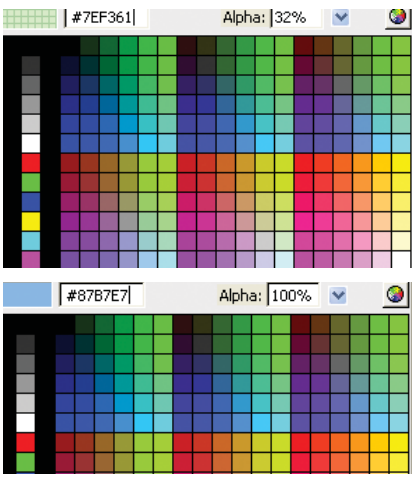
```
("/object", true);
_root.drag = object;
```



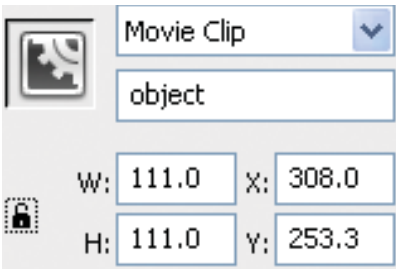
- Now insert a new layer above the Background layer and name it as "Circle".
- Next select the Oval Tool.
- Set the Stroke colour as None.
- Now select any colour for Fill color.



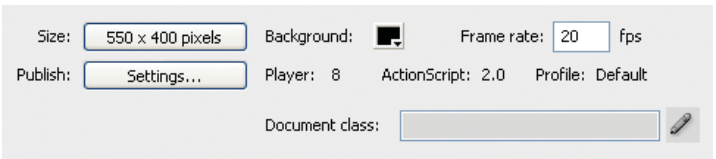
- Next draw a circle over the text.
- Keep the circle selected and open the Color Panel from the Window menu.
- In the Color Panel use these options:
- Select the paint bucket icon to select the Fill colour.
- Select Radial as Type.
- Click on the small colour rectangle situated on the left side.
- Set its colour to light green (#7EF361) and alpha properties to 32%.
- Now click on small colour rectangle on the right side.



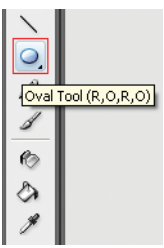
- Here set the colour to light blue (#87B7E7) and alpha properties to 100%.
- You can see that the picture appears somewhat like this.



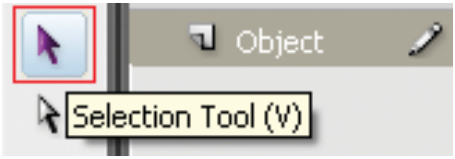
- While the picture is still selected press [F8].
- Type the name as "Circle_mc" and click on Movie clip.
- Keep the Movie clip selected and open the Properties Inspector.
- Type the instance name as 'object'.
- Next select the Text layer and right click on it to choose 'Mask'.

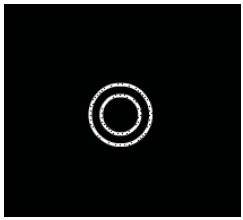


8.7 Mouse Trail

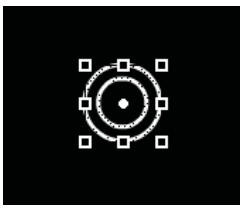


(Note for developer: dimension – 550 x 400 px, Frame Rate – 20 fps, Background color - black)
Let's learn to create mouse trail using the Action Script. So let's start.



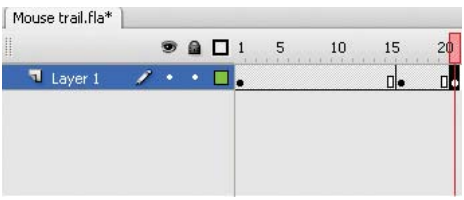


- First rename the layer as “Object”.
- Select the Oval Tool from the Tool box.



- Set the Stroke color as white.
- Now set the Fill color as none.
- Next, draw a circle.
- Click on the Selection Tool.
- Select the circle with the help of this tool.
- Press `[Ctrl] + [C]` to copy the circle.
- Now press `[Shift] + [Ctrl] + [V]` to paste in place the circle.
- Select the Free Transform Tool.
- Slightly decrease the size of the circle. (do it by pressing the Alt key)

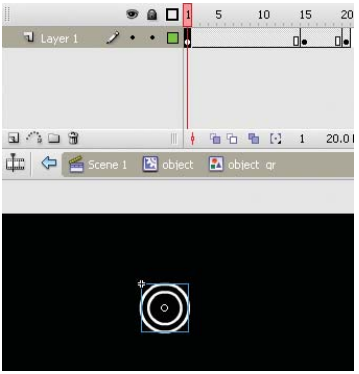
- Now click on the Selection Tool.

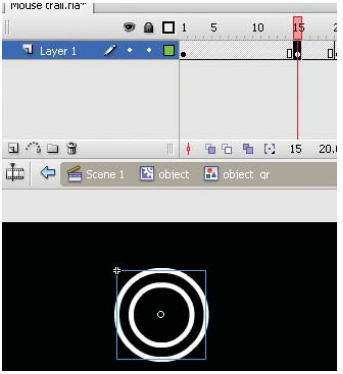


- Click on the Object layer to select the circles.
- Press `[F8]` to convert the circles into movie clip symbol.
- Type the symbol name as “object_mc”.
- Choose Movie clip as symbol type.
- Click on OK.
- Now double click on the movie clip to enter its timeline.
- Keep the circles

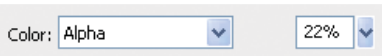
still selected, press [F8] once again.

- Type the name as “object_gr”.



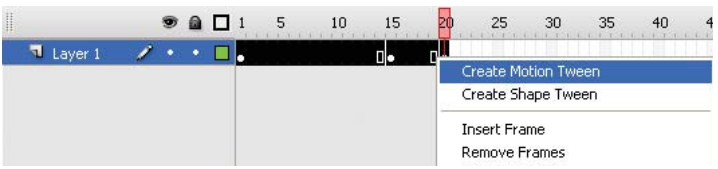


- Choose Graphic as symbol type.



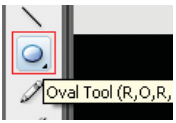
- Click on OK.
- Hold the Ctrl key, click on frame 15 and after that on frame 20.
- Press [F6].
- Then, go back to frame 15.
- Select the Free Transform Tool.
- Increase the size of the circle a little.

- Open the Properties Inspector.
- Choose Alpha under Color option.
- Set the Alpha value as 22%.
- Now click on frame 20.

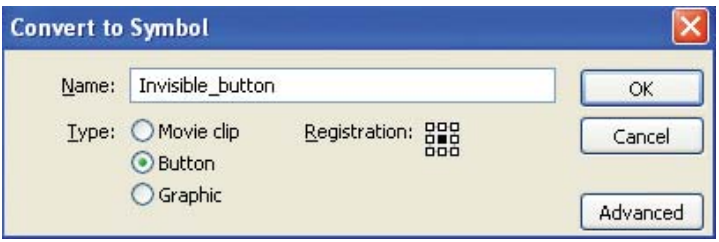


- Click on the object to select it.
- Again open the Properties Inspector.
- Select Alpha from the Color option.

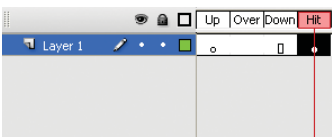
- Set the Alpha value to 0%.
- Now click on the layer.
- Move the mouse cursor on the selected frame.
- Right click and choose Create Motion Tween from the list.
- Now using the drag and drop method, move the whole movie on frame 2.

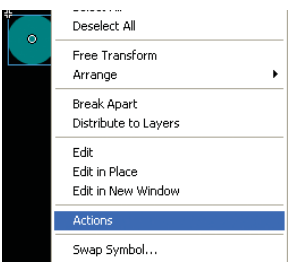


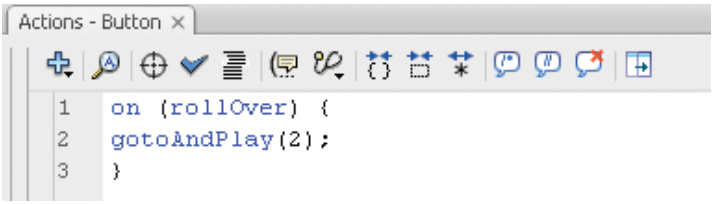
- Insert a new layer.
- Select the Oval Tool.
- Set the Stroke colour as none.



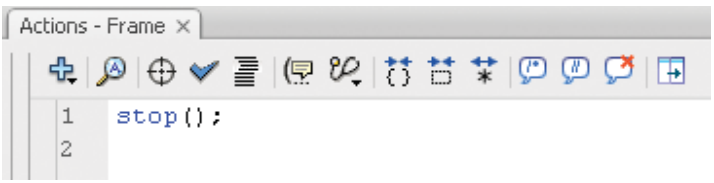
- Select any colour as Fill colour.
- Draw a circle on the same place, where we've placed the object.
- Now select the circle with the help of Selection Tool.
- Press `[F8]`.
- Type the name as "Invisible_button".
- Choose Button as symbol type.
- Click on OK.
- Double click on the button to enter its timeline.
- Drag the keyframe from Up area position to the Hit area.
- Move back to the movie clip's timeline.



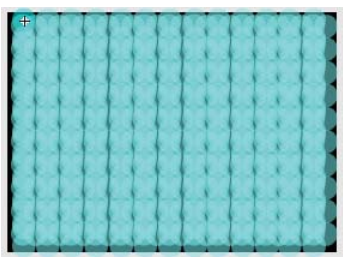




- Right click on the button.
- Choose Actions from the list.
- In the Actions Panel, type this script:



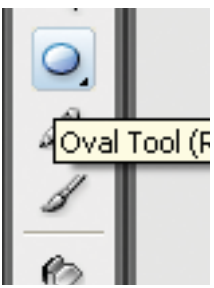

```
on (rollOver) {
        gotoAndPlay(2);
}
```



- Next, click on the 1st frame of the top layer i.e. button layer.
- Once again open the Actions Panel.
- Here, type this script :

```
stop();
```

- Go back to the main scene.
- Duplicate the object many times.
- To duplicate, hold the Shift + Ctrl + Alt keys and click and drag the object.
- The animation is complete now.



8.8 Roll Over Effect

Let's learn to create a roll over effect in Flash.

- You can see a blank document.
- In the Properties Inspector, set the Frame rate to 60 fps.
- Now we will draw a shape.
- Here we draw an oval shape.

- To do this, select the Oval Tool.
- Set the Stroke colour as light blue (#00CCFF).
- Now set the Fill colour as None.
- In the Properties Inspector set the stroke height as 2.

- Click on the Selection Tool.
- Now select the object and press `[F8]`.
- Then convert it to a Movie clip and type the name as "circle".
- Next double click on the new made Movie clip to get inside it.
- Keep the object selected and press `[F8]`.
- This time also we will convert it to a Movie clip but name it as "circle_inside".
- Now open the Actions Script Panel.
- Now add this script.

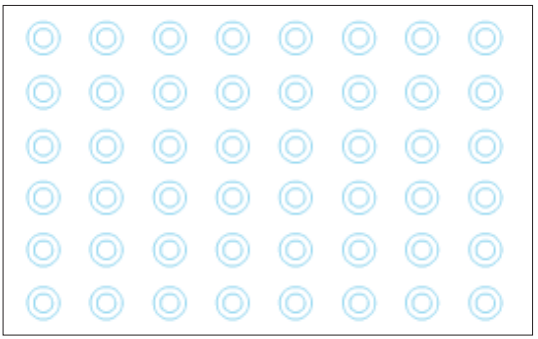
```
onClipEvent (load) {
     baseX = _parent._x;
     baseY = _parent._y;
}
onClipEvent (enterFrame) {
     distanceX = _root._xmouse-_parent._x;
     distanceY = _root._ymouse-_parent._y;
     if (distanceX < 0) {
            distanceX = -distanceX;
     }
```

```
    if (distanceY < 0) {
            distanceY = -distanceY;
    }
    distance =
Math.sqrt((distanceX*distanceX)+(distanceY*distance
Y));
```

Actions - Movie Clip

```
1  onClipEvent (load) {
2      baseX = _parent._x;
3      baseY = _parent._y;
4  }
5  onClipEvent (enterFrame) {
6      distanceX = _root._xmouse-_parent._x;
7      distanceY = _root._ymouse-_parent._y;
8      if (distanceX < 0) {
9          distanceX = -distanceX;
10     }
11     if (distanceY < 0) {
12         distanceY = -distanceY;
13     }
14     distance = Math.sqrt((distanceX*distanceX)+(distanceY*distanceY));
15     if (distance < 350 and distance >-150) {
16         _parent._xscale = distance;
17         _parent._yscale = distance;
18     }
19 }
20 onClipEvent (mouseMove) {
21     updateAfterEvent();
22 }
23
```

```
    if (distance < 350 and distance >-150) {
            _parent._xscale = distance;
            _parent._yscale = distance;
    }
}
```



```
onClipEvent (mouseMove) {
    updateAfterEvent();
}
```

- Keep the object selected and go to the Properties Inspector.
- There choose Alpha for colour and set the value to 45%.
- Now move back to the main scene.
- Copy and paste the object a few times.

8.9 Sparking Effect

Let us learn to create a sparking effect. As you move the mouse you can see some random sparks generating with your mouse movement.

- First we open a Blank Flash File with dimension 550 X 400 pixel
- Set the frame rate to 50
- Set the background to black
- Create a small black circle outside the stage area
- Convert it to a movie clip
- Set the instance name of the movie clip as "spark"
- On the 1st key frame add the following code:

```
for (var i=0; i<100; i++) { (//Duplicate the spark
100 times when the movie loads)
duplicateMovieClip(_root.spark, "spark"+i, i);
(//+I will generate new instance names for each of
the duplicated spark like spark1, spark2 etc)
}
```

- Now right click on the movieclip and add the following script

```
onClipEvent (load) {//sets the spark position to
that of the position of the mouse
    _x=_root._xmouse;
    _y=_root._ymouse;

    //The starting move speeds. random() returns a
float between 0 and 1.
```

```
    var xmove = Math.random()*10 - 5;
    var ymove = Math.random()*10 - 5;
    //frames to wait until begin fading
    var delayuntilfade = Math.floor(Math.ran-
dom()*40-40);
    //create an instance of the Color class
    var thiscolor = new Color(this);
    //just a random float between 0 and 3
    var randcolor = Math.random()*3;
    if(randcolor>2){
          //set the color to red
          thiscolor.setRGB(0xFF0000);
    }
    else if(randcolor>1){
          //set the color to orange
          thiscolor.setRGB(0xFF9900);
    }
    else {
          //set the color to yellow
          thiscolor.setRGB(0xFFFF00);
    }
}
onClipEvent (enterFrame) {
    //decrement the number of frames until fading
begins
    delayuntilfade—;

    //if fully faded, reset to full opacity and
move the spark back to cursor position

    if(delayuntilfade<=(-20)){
          _alpha=100;
          _x=_root._xmouse;
          _y=_root._ymouse;
          //randomize its trajectory
          var xmove = Math.random()*10 - 5;
```

```
            var ymove = Math.random()*20 - 10;
            //reset the frames to wait until fading
again
            delayuntilfade = Math.floor(20+Math.ran-
dom()*50);
    }
    //move the spark
    _x=_x+xmove;
    _y=_y+ymove;
    //gravity
    ymove=ymove+.5;
    //fade the spark %5
    if(delayuntilfade<=0){
            _alpha=_alpha-5;
    }
}
```

Are you drowning in Tech Problems?

Specialist lifeguards are just a click away...

If you have a technology related problem, just log on to thinkdigit.com, visit our Tech Q&A section and let one of our experts solve it for you.

Troubleshooting has never been easier

www.thinkdigit.com

Presents

TECH Q&A